A Framework for Assessing Object-Oriented Analysis Methods

Thesis

UNRESTRICTED

Ying Liang BSc., MSc.

A Framework for Assessing Object-Oriented Analysis Methods

Submitted for Ph.D. in Computer Science

The Open University

October 1995

Date of submission: 6 October 1995
Date of award : 12 March 1996

Abstract

Research into the literature of object-oriented analysis methods finds no evidence of any framework available which might provide a basis for understanding individual object-oriented analysis methods. In order to overcome this deficiency, we establish a framework which focuses on the analysis features of object-oriented methods as well as the logical connections among these features and which enables people to understand object-oriented analysis methods individually by assessing these methods in an objective and systematic way.

The definition of the framework is based upon the study of object-oriented analysis methods available in a wide range. Four representative object-oriented analysis methods are in particular used in a specific application in the study in order to identify the generic features of object-oriented analysis methods. Two aspects (i.e., 'what' and 'how' aspects) of analysis methods are found to be fundamental in the methods and they are emphasised by this framework. The essential features of the methods are therefore identified and assessed upon the two aspects, by means of the framework. Furthermore a process including the approach and criteria is provided for managing the assessment of the methods.

Ten object-oriented analysis methods available have been assessed individually using this framework. The processes of assessing five of the methods are shown in the thesis in detail, as the examples of applying the framework. In addition, the assessment of the other five methods is also outlined and included as it may be useful for people to understand these methods from different perspectives.

Acknowledgements

I am very thankful to

- Mike Newton and Hugh Robinson, my supervisors, for their guidance and discussion on the work in this thesis and for their patience during the write-up,

- the Faculty of Mathematics and Computing at the Open University for offering me a studentship for this study,

- my husband, Yi Fan Xu, for his love, encouragement, and support throughout the study, and

- my parents for their understanding and supporting during the study.

Contents

Chapter 1

Introduction

A variety and range of object-oriented analysis methods are currently available for use in the construction of software systems utilising object-oriented technology. It is important to gain both a comprehension and a deeper understanding of the nature of these individual methods in order to interpret them precisely and correctly. For example, we need to understand why they take the form that they do, what fundamental assumptions they make about object-oriented analysis, to what extent the similarities and differences between object-oriented analysis methods are real rather than merely apparent, and so on. A comprehensive basis that includes the essential aspects of these analysis methods needs to be provided in order to answer such questions. Such a basis would enable individual object-oriented analysis methods to be analysed and assessed. Unfortunately, very little effort has been made so far on the creation of such a basis, as will be demonstrated through our research. Although many comparative studies of current object-oriented analysis methods have been carried out in recent years they can hardly be used as such a basis since they only concentrated on the (comparatively superficial) features that distinguish one analysis method from another, failing to address the deeper and significant fundamental constructs of the methods. Further research into object-oriented analysis methods has to be done in order to provide a basis for genuine comprehension and understanding, without which little real progress will be made in either applying the methods or in enhancing existing methods and in developing new methods.

Historically, a similar change of emphasis — from a comparative study to a more fundamental study — can be found in the research on traditional development methods.

For example, in the early 1980s, a series of comparative reviews of information system analysis and design methods (CRIS) were undertaken by IFIP Working Group 8.1, to obtain the comparative information concerning a range of development methods [Olle82 and 83]. Although many interesting results were produced from these comparative studies that might be helpful in choosing an appropriate method for a desired application [Verrijn82], these studies did not really provide a basis for an understanding of individual information system development methods. To overcome this limitation, a *framework* (that is, a more fundamental basis) was established by the group in the late 1980s [Olle88] which focused on some essential features of these methods.

Our research work aims to establish a similar framework that provides a comprehensive basis for understanding individual object-oriented analysis methods in an objective and systematic way. In this chapter, the background of our research work is briefly given in Section 1.1. The comparative studies of object-oriented analysis methods are outlined in Section 1.2 and their limitations are discussed in Sections 1.3. The aims of our research work are stated in Section 1.4 and, finally, the structure of the thesis is presented in Section 1.5.

1.1 Object-Oriented Analysis Methods

Object-oriented analysis, its role in system development, and object-oriented analysis methods are discussed in this section as the necessary background knowledge to this thesis.

1.1.1 Promotion of Object-Oriented Analysis

Analysing system requirements is a stage in system development in which analysts need to determine what a software system needs to do, according to the system requirements, without concern for implementation details. Analysis can be carried out by using various analysis skills and methods, such as entity-relationship modelling [Chen80 and 83, for

example], structured analysis [DeMarco78, Gane79, Yourdon89, for example] and now object-oriented analysis [Coad91a, Rumbaugh91, for example].

Historically, although it is claimed that rudimentary 'object-oriented' techniques had been even used by the designers of the Minuteman missile as early as 1957 [Ten89], it is commonly accepted that the object idea was introduced in the computer field in 1967 when a discrete event simulation language, *Simula-67,* was created by Kristen Nygaard and Ole-Johan Dahl in Norway [Dahl78, Salmons93]. The term 'object-oriented' initially appeared when the language Smalltalk [Byte81] was developed in the 1970s [Graham94]. Since then more and more programming languages that lay claim to be 'object-oriented', such as C++ [Stroustrup86, Wiener88] and Eiffel [Meyer88], have been promoted by academics and companies in the last few years.

In order to produce a large and complicated software system with object-orientation, since the late 1980's the O-O industry has started to shift its focus, from its former preoccupation with programming languages and issues, to more general concerns about architectural issues including the more elusive continuum involving analysis and design [de Champeaux91, Wiener91]. As Cook [Cook93] states, the development of object-oriented systems as a discipline has been marked by conferences that have taken place since 1986, and object-oriented design appeared initially in 1986 when Grady Booch introduced the concept of object-oriented design to the Ada community. This design approach has been used on a significant percentage of Ada development efforts [Booch86]. More object notation and object-oriented design methods (e.g., [Wasserman89, Ince91, Capretz93]) were developed subsequently [Walker92]. Object-oriented analysis has only been emphasised in recent years [Coad91a, Rumbaugh91, for example]. Prior to the development of object-oriented analysis methods, system requirements were analysed by using conventional analysis methods before designing this system in terms of object-oriented design methods [Jamsa84, Alabiso88, Kennedy88, Ward89, Constantine89, Sully90, Lee91, de Champeaux91, Eckert94].

1.1.2 The Role of Object-Oriented Analysis in System Development

A majority of authors of books and papers indicate that object-oriented analysis and object-oriented design have different concerns and strategies in systems development, object-oriented analysis having, in particular, a distinct content and a distinct role. For example, Coad and Yourdon [Coad91b] state that analysis is the process of extracting the 'needs' of a system—what the system must do to satisfy the client, not how the system would be implemented. A further distinction was made in [Coad91c]: "OOA identifies and defines classes and objects that directly reflect the problem domain and the system's responsibilities within it. OOD identifies and defines additional classes and objects, reflecting an implementation of the requirements. OOA and OOD are distinct disciplines—whether applied in sequence or in some intertwined fashion."

Rumbaugh [Rumbaugh94] distinguishes between analysis and design in the following fashion: "analysis is understanding a problem; design is devising a strategy to solve the problem; implementation is building the solution in a particular medium. This does not mean that these stages are rigidly defined, but I think that they are useful mile posts along the way, just as we distinguish between an outline and a book, although some of the intermediate steps may overlap."

Shlaer and Mellor [Shlaer92] suggest that object-oriented analysis is for identifying the significant entities in a real-world problem and for understanding and explaining how they interact with one another.

Jacobson *et al* [Jacobson92] defined object-oriented analysis as 'in analysis, an application-oriented specification is developed to specify what the system offers its users.'

Henderson-Sellers [Henderson92] differentiates between object-oriented analysis and object-oriented design thus: "the analysis-level model is then primarily concerned with providing an accurate picture of the real-world situation, and an OOA model must have this as its primary objective. The object-oriented design (OOD) model's major objective is to support 'good' software engineering design in terms of correctness, modularity, reusability, and abstraction (Meyer (1988)). ... This separation of analysis and design, and the explicit recognition of language constructs and analysis constructions, are fleeted in,

and supported by, currently emerging analysis and design methodologies, e.g., Coad and Yourdon (1991), Wirfs-Brock *et al* (1990), Booch (1991), Henderson-Sellers and Edwards (1990), and Rumbaugh *et al* (1991)."

According to these authors' points of view, object-oriented analysis and design have different strategies and heuristics — as well as content — in system development, because of their different concerns in system development. Object-oriented analysis emphasises what a software system needs to take into account in accordance with system requirements and creates a specification of a system; whilst object-oriented design concentrates on how to realise this system in a computer environment and how to construct the architecture of this system. In this respect, our distinction between analysis and design is in sympathy with that of Embley, Jackson and Woodfield [Embley95]. These differences are essential in system development, even if some methods may use similar notations to represent both object-oriented analysis results and object-oriented design results.

1.1.3 Object-Oriented Methods

Various object-oriented techniques and methods relevant to object-oriented analysis have been reported on in conferences or published in the literature in the last few years (e.g., [Booch86 & 91, Shlaer88 & 92, Bailin89, Bear90, Helm90, Gibson90, Kirk90, Wirfs89 and 90, Freitas90, Ackroyd91, Coad91a, Lee91, Rumbaugh91, Embley92, Henderson92, Martin92a and 92b, Nerson92, Drake92, Sully93, Honiden93, Coleman94]). Because object-orientation makes it possible to develop and evolve an object-oriented system from the analysis stage to the design stages without any sudden discontinuity [Rumbaugh94], some methods carry out system development without a clear separation between analysis and design (e.g., [Wirfs91]), whilst other methods may separate analysis and design into two clear stages (e.g., [Coad91a and 91b]). For the latter, the first stage of a method is usually described as an "object-oriented analysis method" as only object-oriented analysis

is supported, whilst for the former, a method is usually simply called an "object-oriented design method" even though it also covers the analysis stage of system development.

In different studies, the object-oriented methods that play a dual role in system development (i.e., doing both analysis and design) may be regarded as object-oriented analysis methods (e.g., [de Champeaux92]), or as object-oriented design methods (e.g., [Fowler91]), or as object-oriented analysis-and-design methods (e.g., [Monarchi92, Arnold91]) because of the different emphases of these studies. Our research focuses on object-oriented analysis but not object-oriented design and therefore, for the further purposes of the work, we view them to be object-oriented analysis methods. That is, we call all such methods "object-oriented analysis methods" (or, where the context is clear, simply "analysis methods") in this thesis as long as they support object-oriented analysis in system development.

1.2 Comparative Studies Of Object-Oriented Analysis Methods

Some comparative studies of object-oriented methods have been carried out, concentrating mainly on superficial similarities and differences between methods. The studies examined in our research included [Arnold91], [de Champeaux92], [Fowler91] and [Monarch i92]. They are briefly described as follows.

(1) The Comparative Study of Arnold [Arnold91]

— Objective

To highlight the similarities and differences between analysis and design methods in order to assist the practitioner in deciding on an appropriate method for their applications.

— Focus

Object-oriented analysis and design.

— Comparative features

The comparison is carried out by scoring methods against a set of possible features that cover four main aspects of OOA and design methods:

- *concepts* that the methods may use,
- *models* that the methods may provide,
- *processes* that the methods may offer for building the models, and
- *pragmatics* that the methods may address as non-technical features.

These features are emphasised in the study since the authors assume that (a) a disciplined software process is the essential factor determining success [Coleman91], (b) a method should support the representation of the concepts that assume the most prominent role in object-oriented systems, (c) a method proceeds by developing models of systems under analysis or design, and (d) other factors that may influence a method's usage in the software engineering community.

(2) The Comparative Study of Fowler [Fowler91]

— Objective

To provide a way of looking at what is available for analysis and design, and what are the similarities and differences between methods.

—Focus

Object-oriented analysis and design.

—Comparative features

This comparison is only concerned with one feature that object-oriented methods are likely to have; i.e., object-oriented modelling, with three different views of systems:

- *data* view,
- *behavioural* view, or
- *architectural* view.

This feature is focused on since the author supposes that (a) the view of systems addressed by a method implies a way to model applications, and (b) different analysis and design methods may analyse the same application in terms of different views.

(3) The Comparative Study of de Champeaux [de Champeaux92]

— Objective

To describe what analysis methods have in common and where they differ.

— Focus

Object-oriented analysis.

— Comparative features

The comparison is concerned with the following features:

- *Purpose* of the methods,
- *Models* offered by the methods,
- *Boundary* of analysis and design,
- *Graphic notations*,
- *Relationships* among objects and classes,
- *Object creation and destruction*, and
- *Process* of analysis.

These comparative features are selected since the authors assume that these features are shared by a majority of object-oriented analysis methods.

(4) The Comparative Study of Monarchi [Monarchi92]

— Objective

To provide a framework for comparing and evaluating current analysis and design methods.

—Focus

Object-oriented analysis and design.

—Comparative features

Three critical features are included in this framework:

- *Representation* offered by the methods,
- *Process* of analysis or design supported by the methods, and
- *Complexity management*.

These features are selected from the analysis and design characteristics described by Colter [Colter84] and Pressman [Pressman87].

1.3 Limitations of the Comparative Studies

The comparative studies introduced in the previous section focus on different object-oriented methods that support either object-oriented analysis or object-oriented analysis-and-design in system development. The studies produced a list of the (mainly superficial) similarities and differences between the methods. However very little exploration of the extent to which these similarities and differences were real rather than apparent was made in these studies since their objectives did not address such issues. In particular, the major limitations of these studies to an understanding of object-oriented analysis methods are as follows.

1.3.1 No Separation of Analysis Features and Design Features

These comparative studies did not emphasise the distinction between analysis features and design features. Extra efforts have therefore to be made in order to explore the analysis features and the foundations of object-oriented analysis methods.

Section 1.1.2 has shown that object-oriented analysis is fundamentally different from object-oriented design since they have different strategies, heuristics and concerns in system development. Consequently, the features and contents of analysis and design are also different. For example, all analysis methods (e.g., [Coad91a] and [Rumbaugh91])— but not object-oriented design methods (e.g., [Coad91b]) — have a feature that identifies objects from the system requirements, as analysis is concerned with problem domains rather than computer domains [Booch91]. A clear separation of analysis features from design features is essential in order to analyse the strategies, heuristics and concerns of individual analysis methods in system development, to reveal the particular and fundamental characteristics of those methods, and to understand them more objectively, precisely and correctly.

1.3.2 No Assumption of Logical Connections between Comparative Features

The features of an analysis method may have an impact on one another to some extent because of the inherent logical connections between them in the method. One typical example is the impact of an analysis model on the analysis process in a method. The analysis process ought to cover the definition of each element in an analysis model since the process must satisfy the detail of constructing the object models. The OOA method [Coad91a], for instance, defines an object model as five-layer model and therefore it provides five analysis activities in its analysis process; while the OMT method [Rumbaugh91] creates three different models in analysis and therefore three major steps are included in its analysis process, each step being responsible for constructing one model.

A consideration of the logical connections between the features of an analysis method is essential in order to understand the method from different viewpoints and to explore the details and motivations behind the features (for instance, why a method has particular features and to what extent these features are supportive of each other). Comparative studies, by their nature, do not emphasise the logical connections between the comparative features.

1.3.3 No Emphasis on the Understanding of Object-Oriented Analysis Methods

In view of the objectives of these comparative studies, they do not address any understanding or comprehension of individual analysis methods; i.e., issues such as why these methods take the form that they do, what the fundamental constructs of the methods are, why methods make different assumptions about both the process of analysis and the nature of object-oriented technology, and so on. The studies are therefore not appropriate as a comprehensive basis for understanding any individual analysis method. Although the studies bring out the (superficial) similarities and differences between analysis methods,

they do not clarify to what extent these similarities and differences are real rather than apparent. For example, the term 'object' has different meanings in the OOA method [Coad91a] from that of the Wirfs-Brock method [Wirfs90]: the former assumes that an object is an encapsulation of attribute values and their exclusive services that act on the attributes; while the latter supposes that an object encapsulates both functions and data that are included in the functions. Thus these two methods seem to have the apparently similar feature 'object', but this is not true in terms of the real content of each one. The comparative studies do not focus on and explore such detail, in particular, they do not explain why the same term has different meanings in two, or more, methods.

1.4 Aims of Our Research Work

At the beginning of Chapter 1, we mentioned that, for understanding traditional development methods, a framework had been established by IFIP Working Group 8.1 [Olle88], based upon the comparative study and feature analysis of the methods [Olle82 and 83]. To understand traditional analysis, the framework covered the features which were regarded as fundamental in the methods: *modelling concept, model with different perspectives of a system,* and *deliverable to design*. We, however, do not think that it is appropriate to use Olle's framework as a comprehensive basis for understanding object-oriented analysis methods because of the following reasons:

(i) Olle's framework is not concerned with object-orientation and thus it does not emphasise object concepts and models.

(ii) There is a debate as to the compatibility of object-oriented development with traditional analysis [Firesmith91, Shumate91]. The concepts of traditional analysis may not be appropriate for adoption in object-oriented analysis.

(iii) The study of current object-oriented analysis methods shows that some concepts such as 'inheritance relationship' and 'object behaviour' are new to system analysis

and modelling. This means, it is impossible to use Olle's framework for assessing and understanding such concepts in object-oriented analysis methods.

A new framework that focuses on object concepts and which supports understanding of object-oriented analysis methods has, therefore, to be established. Although Olle's framework cannot be used to understand object-oriented analysis methods, the process of developing the framework, as illustrated in [Olle82, 83, 88], provides a useful experience for establishing a similar framework. That is, to build a framework for understanding object-oriented analysis methods, the first step is to study existing object-oriented analysis methods, e.g., by using the methods in applications or by textbook research, so that the essential features of the methods are explored naturally; and the second step is to identify and capture the essential features of the methods discovered in the first step and then define them in the framework. Our research aims to establish the required framework in a similar process, including a rectification of the limitations in the comparative studies of object-oriented analysis methods.

1.4.1 A Framework for Assessing Object-Oriented Analysis Methods

As a basis for understanding the nature of individual object-oriented analysis methods, the proposed framework covers a set of basic features which a majority of these analysis methods have. This framework however never assumes which method is best, instead it aims to clarify the meanings and the contents of these features in an individual analysis method. Unlike Olle's framework, which reflects the standards of traditional modelling (e.g., 'data flow', 'process', 'entity', 'entity relationship', 'event', 'state', and so on), our framework does not emphasise any standards in object modelling, since object-oriented analysis methods are still immature and it is difficult to choose and decide on standards at present.

This framework enables a method to be 'disassembled' into small (and fundamental) parts so that it can be analysed and assessed more easily. The logical connections between features are also emphasised and expressed in terms of the framework, to provide a deeper

understanding of a method. Another expected contribution of the framework is that it supports a broader, objective appreciation of what an object-oriented analysis method ought to contribute to system development. In general, this framework can be regarded as a vehicle for understanding, innovating, improving, comparing and evaluating analysis methods.

In order to use the framework to assess an analysis method, a process of assessment is provided in section 3.5. In the process, an approach to (and relevant criteria for) using the framework to assess an analysis method are given, so that an assessment of a method can be carried out step by step, ensuring that the features identified are consistent with the components in the framework. The criteria are specified in a questionnaire form so that the features — and the relationships between the features — are identified by answering the questions.

1.4.2 A Study of Object-Oriented Analysis Methods

As stated above, in a similar fashion to that of Olle's framework, our framework is built upon a wide study of many existing object-oriented analysis methods. The first step in our study is to focus on obtaining both theoretical knowledge and practical experience about the methods. The theoretical knowledge about the methods comes from reading the text books which introduce and interpret the methods; while the practice experience is gained by using typical analysis methods in a specific application so that it can be found whether or not the essential characteristics of object-oriented analysis methods, claimed in the text books, are really supported by — and useful to — their application.

In this study, four representative methods [Coad91a, Rumbaugh91, Booch91, Wirfs90] are particularly investigated both by reading the text books to see what the methods promise to do, and by using them to analyse a specific problem — a book trader scenario— in order to explore the extent to which these promises are realised in practice. The use of the methods may provide more details behind the textual description. The study focuses on the following aspects:

- *what each method promises to do for analysis,*
- *how each method carries out analysis,*
- *what (and why) assumptions are made by each method for analysis, and*
- *what (and how) analysis product is generated by each method.*

This study enables us to recognise the fundamental assumptions about object-oriented analysis in general, providing a basis for identifying and determining the essential features in analysis methods. These essential features are incorporated into the framework, since they are considered significant in the understanding the nature of object-oriented analysis methods.

1.4.3 Applications of the Framework

The framework established by this study enables us to assess any method individually, using the process of assessment provided in section 3.5. Ten methods have been assessed by using the framework in our research. Because of the size limitation, however, this thesis only presents the assessment process and the results for five representative object-oriented analysis methods (i.e., the OOA method [Coad91a], the OMT method [Rumbaugh91], the Booch method [Booch91], the Wirfs-Brock method[Wirfs90], and Syntropy [Cook94a]) in detail (in Chapter 4 and 5). In addition, the assessment results of another five methods (i.e., the Shlaer and Mellor method [Shlaer88], OOSE [Jacobson92], OSA [Embley92], Ptech [Martin92a] and HOOD [Robinson92]) are outlined in the Appendix A. Nevertheless, these applications of the framework provide useful examples and, in particular, demonstrate the process of using the framework in the assessment of any other object-oriented analysis method.

It should be noted that we have so far already published two papers, [Liang93 and 94], which show the work completed in the early stage of our research. These two papers have also provided a basis for establishing the framework presented in this thesis.

1.5 The Structure of the Thesis

As stated in the previous section, our research work includes three major parts: a study of a number of methods — in particular, four methods which were used for a specific application; the establishment of the required framework (based on the study of the methods); and the applications of this framework. The presentation structure of this thesis is thus as follows: the study of four representative methods and the conclusions of the study are discussed in Chapter 2; the required framework is defined and interpreted in Chapter 3; and the application of this framework to the five methods (the four representative methods plus Syntropy) are demonstrated in Chapter 4 and 5. The conclusions of our research work, the other potential use of the framework, and future work are discussed and indicated in Chapter 6. Finally, the Appendix outlines another five methods assessed by means of the framework.

Chapter 2

A Study of Four Analysis Methods

This chapter describes a study of four analysis methods, i.e., the OOA method [Coad91a], the OMT method [Rumbaugh91], the Booch method [Booch91] and the Wirfs-Brock method [Wirfs90]. These methods are commonly considered as representative methods which are used in many academic and industrial applications [Glykas93], and are also taught as typical object-oriented methods in education [Lovegrove92]. For this study, first we reviewed the text book describing each method and established the claims made by the method, and second we analysed each method in detail, i.e., what the method assumes about analysis, by applying each one to analyse the book trader scenario described in Section 2.1. The detail of the study is given in Section 2.2. Based on the results from the study, the basic assumptions of these methods about analysis are summarised in Section 2.3.

2.1 A Book Trader Scenario

An International Standard, [ISO87], has used an example in a similar way, and recommends that such a Universe of Discourse should be sufficiently small so that a large description is not required, but sufficiently complex to exhibit the essential differences among the various methods. Consistent with this recommendation, we selected a book trader scenario as an example application that is small but adequately complex to exhibit the details such as objects and classes and their attributes and operations, etc. that appear

in many object-oriented methods [Synder93]. In particular, this scenario represents requirements typical of information systems. This enabled us to establish more precisely the assumptions of analysis methods, since it is considered that there are many similarities between object modelling and information modelling. For instance, D. Coleman and F. Hayes said in their paper [Coleman91] that 'the information modelling ideas are especially useful in determining an appropriate class structure'. The OOA method [Coad91a] is even claimed as an integration of object concepts and information modelling concepts. The book trader scenario we used is as follows:

The company is concerned with selling books by advertising in magazines. Each advertisement is placed in one magazine on a given date, and includes a number of books described by their titles, authors and publisher, together with the price at which the company is offering each book; a book may have a different price in different advertisements.

A potential customer can buy one or more of the advertised books by mailing the company an order form, which is included as part of each advert, giving details of their name and address, the title and the price as specified in the advertisement of each of the books they require and the total cost. Payment is required, either by including a cheque for the total amount due or by giving a credit card number. An alternative way of buying books is to telephone the company and give the same order details, but in this case a credit card number is the only way of paying. The company then either waits for a cheque to be cleared by its bank or, for a credit card where the amount due is more than its floor limit, gets authorisation from the card company. Once payment for an order is approved, a receipt is produced, the ordered books are taken from stock and sent to the customer with the receipt.

This business is supported by an information system in which the unique ISBN for each book is recorded, together with its title, the names of its authors, the name of its publisher, its cost price and its current stock level (i.e., the number of copies of the book in the company's warehouse). For each advertisement, the name of the magazine in which it appears and the date of the issue is recorded, as well as the list of books and their price given in the advertisement. Each order is assigned a distinguishing order number and details are recorded of the customer, the date, the books required, the total cost and a credit card number if that is the chosen method of payment.

Customer details include their name and address, but they are only relevant to a specific order (i.e., there is no requirement to associate a customer with all orders they may have placed).

The information system is required to support the above business activities, together with two kinds of enquiries. One enquiry requests the number of orders resulting from a given advertisement. The other enquiry requests details of the progress of a specific order (e.g., Are the books available? Have they been sent? etc.), identifying the order by giving either an order number or the details of the customer and the advert to which they responded.

2.2 The Study of Four Analysis Methods

This section presents the study of the four selected analysis methods by describing the use of each of them to analyse the book trader scenario given above. The following issues in particular were addressed by this study:

(1) what each method claims about object-oriented analysis;

(2) how each method analyses the book trader scenario; and

(3) what assumptions each method makes about analysis.

The study therefore focuses on the following aspects of each method:

(1) claims of the method,

(2) required inputs to the analysis,

(3) analysis addressed, and

(4) products generated from the analysis.

2.2.1 The Study of the OOA Method (Coad and Yourdon)

2.2.1.1 Claims of the OOA Method

The OOA (Object-Oriented Analysis) method [Coad91a] is claimed as an object-oriented analysis method that merges the *best* concepts from information modelling, object-oriented programming languages and knowledge-based systems. It is based on the assumption that 'object-oriented' means 'classes and objects, inheritance and communication with

message'. In particular it focuses on the principles: *abstraction, encapsulation (also information hiding), inheritance, communication with messages, pervading methods of organisation,* and *scale.* An OOA *model* is defined by this method in order to specify the results of analysis. Underlying these principles, an OOA model consists of five layers: class-&-objects layer, structure layer, subject layer, attribute layer, and service layer. To build an OOA model, a data-driven analysis process is provided that includes five activities:

- *Finding class-&-objects, identifying structures, identifying subjects, defining attributes, and defining services.*

These activities can be carried out in any order. The analysis supported by this method is supposed to focus on the problem domain and the system's responsibilities.

2.2.1.2 Inputs Required

The required input to analysis is a description of a problem domain given by users and representing requirements for a system.

2.2.1.3 Analysis Addressed

The claims of the OOA method above show that this method is considered to support analysis of a problem domain (the book trader scenario here) by carrying out the five activities. Our use of this method to analyse the given book trader scenario is as follows.

Activity 1: Finding Class-&-Objects

This activity builds a class-&-objects layer for a book trader system.

Object—An *abstraction* of something in a problem domain, reflecting the capabilities of a system to keep information about it, interact with it, or both; an *encapsulation* of attribute values and their exclusive services. (synonym: an Instance)

Class—A description of one or more objects with a uniform set of attribute and services, including a description of how to create new objects in the class.

Class-&-objects—A term meaning '*a class and the objects in that class*'.

(1) Where to Look

To identify candidate class-&-objects from the book trader scenario, the method advises us to investigate the relevant problem domain by observing first-hand, listening to problem domain experts, checking previous analysis results and other systems, repeatedly reading description, and prototyping the analysis.

(2) What to Look for

We are advised to look for structures, other systems, devices, things or events remembered, roles played, operational procedures, sites, and organisational units from the book trader scenario and then to select class-&-objects that should be significant in the book trader system according to the criteria given by the method.

(3) What to Consider and Challenge

The given criteria that determine the significant class-&-objects are that there should be a) needed remembrance, b) needed behaviour, c) with multiple attributes, d) with more than one object instance, e) with always-applicable attributes, f) with always-applicable services, g) from domain-based requirements, and h) not merely derived results.

From this activity, five class-&-objects— *Book, Advert, MailOrder, PhoneOrder* and *Customer*— were defined in the class-&-objects layer, as shown in Figure 2.1, since they are needed to be remembered or to play a role in a book trader system. The classification of class-&-objects *MailOrder* and *PhoneOrder* was however found difficult since it was not quite appropriate to consider them as things. The names were given based on the *name* rule of OOA: the name of a class-&-objects should be a word in the standard vocabulary of the problem domain.

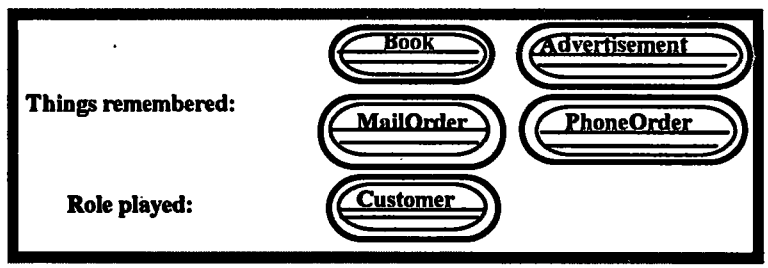

Figure 2.1 The Class-&-Objects Layer of the System

Activity 2: Identifying Structures

This activity identifies generalisation-specialisation and whole-part structures.

> Structure—*Structure* is an expression of problem-domain complexity, pertinent to the system's responsibilities. The term 'structure' is used as an overall term, describing both generalisation-specialisation (Gen-Spec) Structure and whole-part structure.

Generalisation-specialisation (Gen-Spec) structure is considered as a *'is a kind of'* structure among classes: a specialisation class is a kind of generalisation class and it inherits the definition of the latter. *Whole-part structure* is considered as a *'has a'* structure among class-&-objects: a whole class-&-objects has a part class-&-objects. A structure is supposed to help to understand the system's responsibilities and to identify missed objects and classes.

(1) Identifying Generalisation-Specialisation Structures

In this activity, we are advised to consider each identified class-&-objects as either a generalisation or a specialisation, and ask the questions: "Is it in the problem domain? Is it within the system's responsibilities? Will there be inheritance? Will the specialisation's or generalisations meet the 'what to consider and challenge' criteria given in activity 1 for class?". Under this guidance, one generalisation class *Order* was defined on the classes *MailOrder* and *PhoneOrder*. *Order* is a class rather than a class-&-objects, since it is not a description for which there will be objects. This generalisation-specialisation structure is represented by Figure 2.2(a).

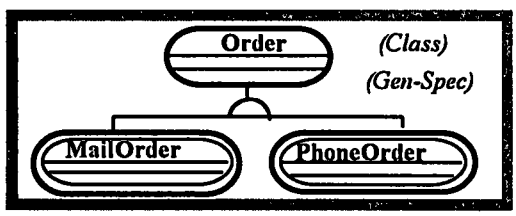

Figure 2.2(a) The Gen-Spec Structure of the System

(2) Identifying Whole-Part Structures

a) What to look for

A potential whole-part structure may be one of the structures: *assembly-parts* (e.g., an aircraft and engine(s)), *container-contents* (e.g., an aircraft and pilots), or *collection-*

members (e.g., an organisation and clerks). One container-contents structure, *Order-Customer*, was defined for the book trader system and is shown in Figure 2.2(b). This choice was based on the guidance that if the problem domain and system responsibilities include a qualified content class-&-objects to a specific container object, a container-contents structure is needed.

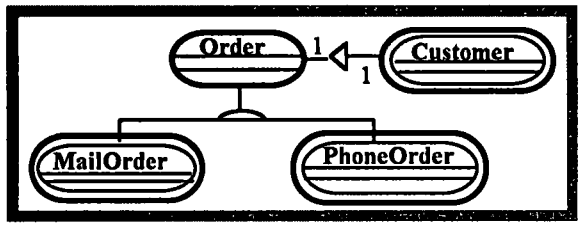

Figure 2.2(b) The Structure Layer of Book Trader System

b) What to Consider and Challenge

In a container-contents structure, each class-&-objects satisfies the criteria given, i.e., it a) is in the problem domain, b) is within the system's responsibilities, c) captures more than just a status value, and d) provides a useful abstraction in dealing with the problem. The container-contents structure *Order-Customer* satisfies these criteria.

Activity 3: Identifying Subjects
This activity builds a subject layer for a system.

> Subject—A *subject* is a mechanism for guiding a reader through a large, complex model. Subjects are also helpful for organising work packages on larger projects, based upon initial OOA investigations.

(1) How to Select

In order to build a subject layer, *uppermost* class-&-objects or classes in structures are promoted as subjects, each of which covers a sub-domain of the problem domain. The class-&-objects *Advertisement* and the class *Order*, in the structure layer of the book trader system were thus promoted.

(2) How to Refine

A subject is refined according to sub-domain, with minimal inter-dependencies (i.e., structures, instance connections) and minimal interactions (i.e., message connections) between it and other subjects. Three subjects were thereby defined for the book trader

system: a) 'advertisement' subject with *Advertisement*; b) 'order' subject with *Order*, *MailOrder*, *PhoneOrder*, and *Customer*; and c) 'book' subject with *Book*, as shown in Figure 2.3.

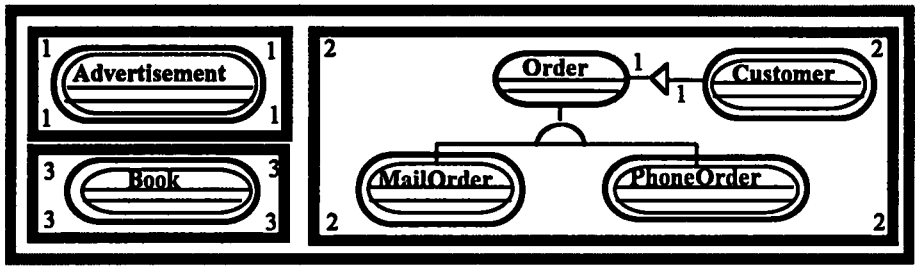

Figure 2.3 The Subject Layer of the System

Activity 4: Defining Attributes

This activity defines an attribute layer for a system.

> Attribute—An *attribute* is some data (state information) for which each object in a class has its own value.

An attribute layer contains the attributes in, and instance connection between, objects.

> InstanceConnection—An *instance connection* is a model of problem domain mapping(s) that one object needs with other objects, in order to fulfil its responsibilities.

(1) Identifying Attributes

The method advises that attributes are identified by asking the questions for a single object: "How is the object described in general? How is the object described in this problem domain? How is the object described in the context of this system's responsibilities (i.e., what does the system need to know about the object)? What does the object need to know? What state information does the object need to remember over time?" Each attribute ought to represent a single value or tightly-related grouping of values. Normalisation and identification mechanisms (i.e., visible identifier) are deferred to design. Each selected attribute is assigned to the class-&-objects or class that it best describes.

Under these guidelines, the following potential attributes were identified and put in the class-&-objects in the book trader system:

Book—ISBN, Title, Author, Publisher, Price, StockLevel

Advertisement—MagazineName, IssueDate
Order—OrderNo, OrderDate, OrderState
MailOrder—PaymentDetail
PhoneOrder—CreditCardDetail
Customer—Name, Address.

(2) Defining Instance Connections

The following *guidelines* are given to identify and define the instance connections among objects within a system:

a) for each object within the system, connection lines are added between it and other objects if they reflect mappings within the problem domain and the system's responsibilities;

b) for a Gen-Spec structure, the connection line is connected with the uppermost applicable level of the structure;

c) the amount or range of the connections, such as one-to-one, is defined for each connection in form of lower bound-to-upper bound;

d) if an upper bound is more than one, check whether connected objects have special constraints (e.g., the most recent time). If so, additional attributes (e.g., DateTime) need to be added to the corresponding class-&-objects; and

e) if a constraint is across more than one instance connection, it can be specified within a connected class-&-objects specification template as 'additional constraint'.

Under these guidelines, three instance connections were identified and defined in the book trader system, as shown in Figure 2.4(a).

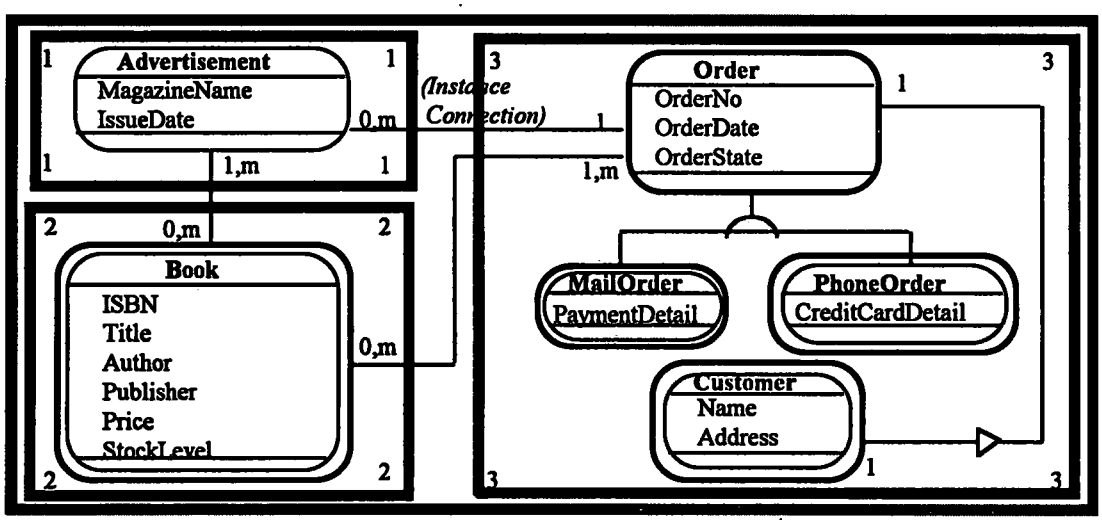

Figure 2.4(a) The *Initial* Attribute Layer of the System

(3) Checking Special Cases

In this action, the initial attribute layer of the book trader system is validated and revised if needed. A *check-list* for attributes and instance connections is as follows:

• For the *attributes* identified, the following aspects need to be considered:

a) for each attribute, if one of its values is *not applicable* or it *does not apply* here, to consider whether another Gen-Spec structure is defined;

b) if a class or a class-&-objects only has *one attribute*, to consider whether or not it is important to the system. If not, it might be defined as an attribute of another class-&-objects; and

c) if an attribute has repeating values, it may be defined as a new class-&-objects.

According to these criteria, the attributes in Figure 2.4(a) were checked as follows:

— According to the book trader scenario, the value of attribute 'PaymentDetail' is either cheque detail or credit card detail, namely there always is one value that is not applicable. According to a), an additional Gen-Spec structure was added to the structure layer of the system in Figure 2.4(a)(see Figure 2.4(b)).

— According to b), there are two class-&-objects, *MailOrder* and *PhoneOrder*, that have only one attribute. They however should be kept since they are important to the book trader system.

— According to the book trader scenario, the price of a book may be different in different advertisements, that is, the value of the attribute 'price' in the class-&-objects *Book* has repeating values with costing price and selling price. According to c), an additional class-&-objects *SellingPrice* was defined with an attribute 'Amount'. This class-&-objects is related to both *Advertisement* and *Book*.

• For *instance connections* between objects, the following aspects need to be validated:

a) for each many-to-many instance collection, in order to describe the connection some attributes might be defined by introducing another class-&-objects;

b) if there exists an instance collection between objects of the same class, it should be reviewed;

c) for multiple instance connections between same two objects, it should be reviewed to determine if an additional class-&-objects may be added between them;

d) for each pair of objects, check the existing instance connections to see if additional ones may need to be added; and

e) if one connecting object has a special meaning, an attribute may be needed to add to the affected class-&-objects.

According to these criteria for instance connections, there was no need to change the book trader system.

The changes to attributes are given in a second version of the attribute layer of the book trader system in Figure 2.4(b). In this version, it was considered that one of the class-&-objects *PhoneOrder* and *CreditCardOrder* should be deleted as their attributes are the same. As a possible solution, we deleted the class-&-objects *PhoneOrder* and added an attribute 'OrderType' in the class-&-objects *Order* to distinguish between mail orders and phone orders. The third version of the attribute layer is shown in Figure 2.4(c).

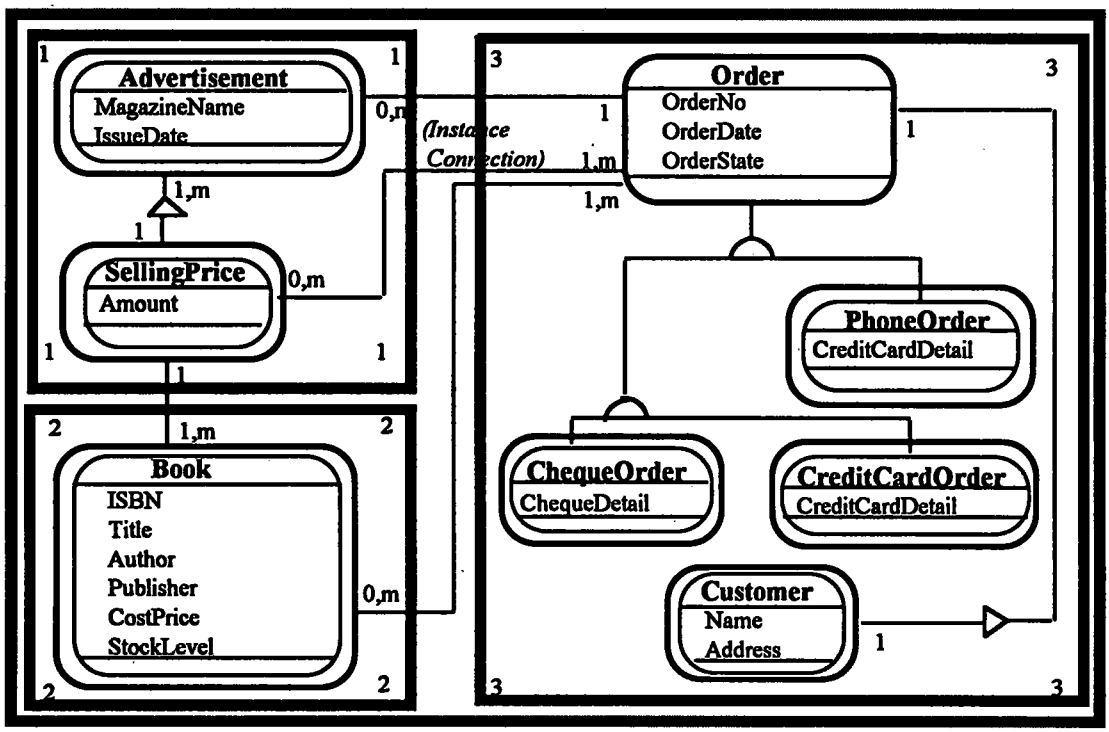

Figure 2.4(b) The Second Version of the Attribute Layer of the System

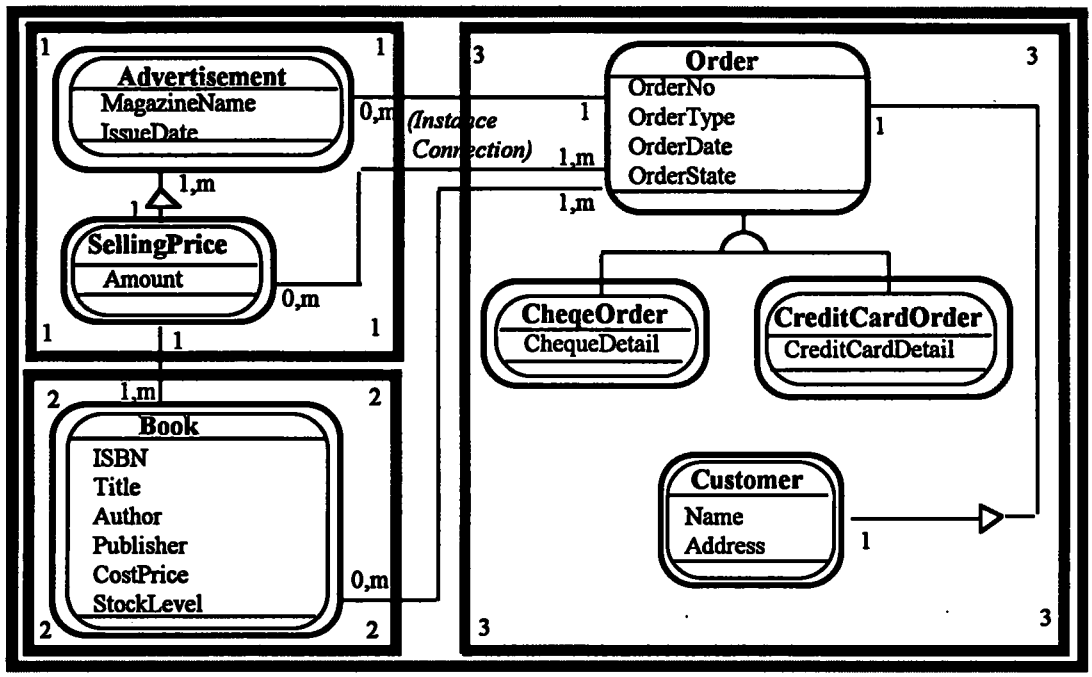

Figure 2.4(c) The Third Version Of the Attribute Layer of the System

Activity 5: Defining Services

This activity describes the services in objects and classes.

Service—A service is a specific behaviour that an object is responsible for exhibiting.

The services in an object are identified by considering its states, required services on it, and the message connection between it and other objects.

(1) Identifying Object States

Object states imply the behaviour of an object from creation to deletion and they are based on the values of the attributes of the object. The states are identified by examining the potential attribute values and then checking whether or not the system's responsibilities include different behaviour for these values. The states and state changes of an object are illustrated by an *object state diagram*.

In the book trader system, *CheckOrder* and *CreditCardOrder* have different states depending on the value of the attribute 'OrderState'; their object state diagrams are drawn in Figure 2.5.

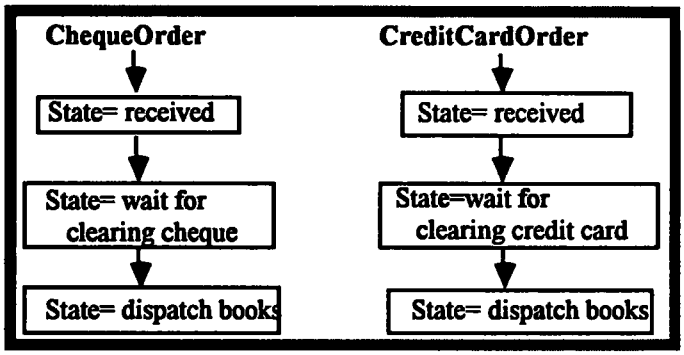

Figure 2.5 Object State Diagrams within the System

(2) Identifying Required Services

Two kinds of services are identified for the OOA method: algorithmically-simple services and algorithmically-complex services.

a) Identify Algorithmically-Simple Services

Usually, four algorithmically-simple services need to be defined for each class-&-objects:

- *Create*, a service to create and initiate a new object in the class,
- *Connect*, a service to establish a mapping between an object with another,
- *Access*, a service to get or set the attribute values of an object, and
- *Release*, a service to disconnect and delete an object.

However, these services do not need to be exhibited in the service layer as every class-&-objects in a system always implies them.

b) Identify Algorithmically-Complex Services

These services are either calculation services or monitoring services:

- *Calculate*, a service to calculate a result from the attribute values of an object, and
- *Monitor*, a service to receive inputs from and send outputs to outside of the system, or to deal with device data acquisition and control.

In order to identify these services, the method advises consideration of: "What calculations need to perform on the attribute values of an object? What monitoring needs to be carried out by the object?" The following algorithmically-complex services were thus identified and put in the class-&-objects and class in the book trader system:

Advertisement—CalculateTotalOrders, TellTotalOrders
Book—ModifyStockLevel
Order—CalculateTotalCost, TellOrderState, ProduceReceipt
ChequeOrder—ClearCheque
CreditCardOrder—ClearCreditCard.

(3) Identifying Message Connections

Identifying message connections is based on the processing dependencies between objects in a system.

> Message Connection—A *message connection* models the processing dependency of an object, indicating a need for services in order to fulfil its responsibilities.

In a message connection, a 'sender' sends a message to a 'receiver'. The required processing is named in the sender's service specification and it is defined by the receiver's service specification. A message connection combines the event-response and data flow perspectives together since each message connection represents the values sent to the 'receiver' and a response received by the 'sender'.

For each object, the method suggests thinking about: "What other objects need the services defined by this object (draw an arrow to each of these objects)? What other objects define the services needed by this object (draw an arrow from each of these objects to this object)?" More message connections may be identified by tracing each message connection defined to the next object.

Following these guidelines, four message connections from *Order* to other objects were identified for the book trader system, as shown in Figure 2.6: a message to *Advertisement* to get the number of orders, a message to *Book* to modify the stock level in it, and the messages to *SellingPrice* and *Customer* to access the information recorded in them.

(4) Specifying Services

Services are specified by *service charts* that define the algorithms of the services. Figure 2.7 shows an example of a service chart that specifies the service 'ClearCheque' within the class-&-objects *ChequeOrder*.

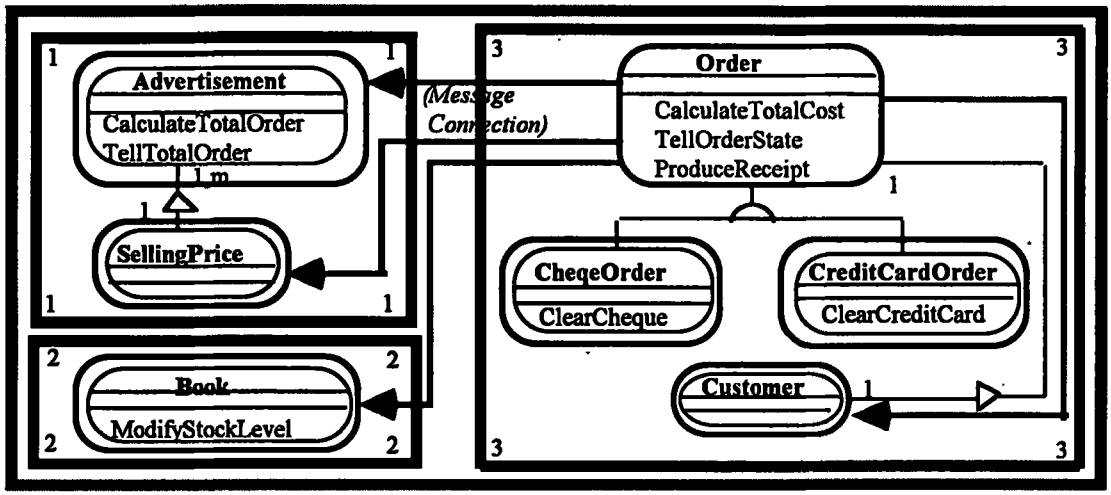

Figure 2.6 The Service Layer of the System

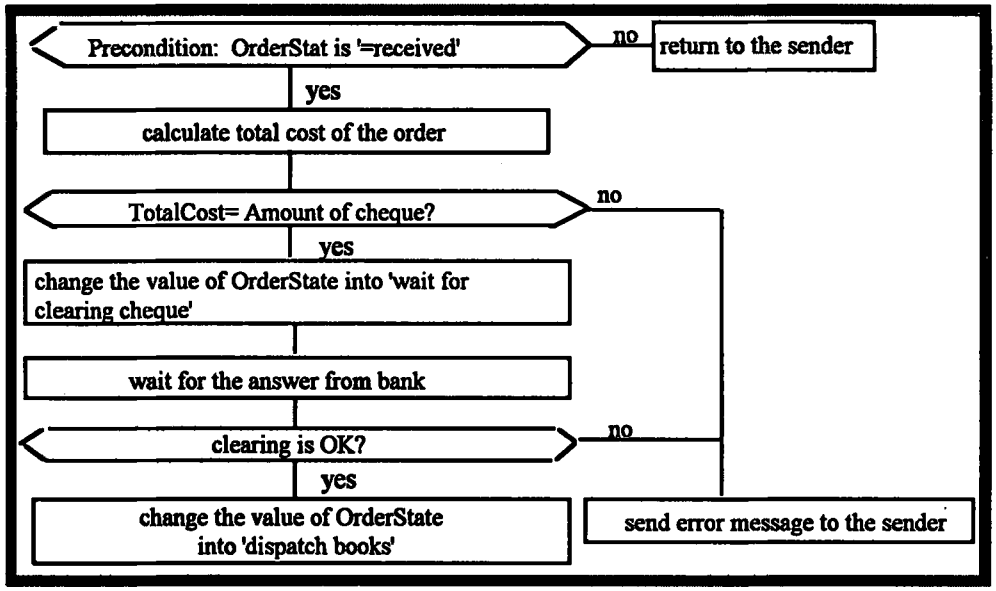

Figure 2.7 A Service Chart for the Service 'ClearCheque'

The full details of each class-&-objects, like *ChequeOrder*, are specified by filling a template as follows:

Specification ChequeOrder
attribute ChequeDetail: cheque number, branch code, etc.
attribute OrderState: received, wait for clearing cheque, dispatch books
externalInput
ChequeDetail: cheque number, branch code, etc.
externalOutput
Progress of the cheque order: the current state of the order
ObjectStateDiagram in Figure 2.5
additionalConstraints
The cheque order cannot be provided by telephone.
service ClearCheque (out: result)
see the *Service Chart* in Figure 2.7
service TellOrderState (out: result)

2.2.1.4 Products Generated

The products from the analysis of the book trader system is an OOA model that consists of five layers together with class-&-objects specifications and two object state diagrams and service charts.

2.2.2 The Study of the OMT Method (Rumbaugh et al.)

2.2.2.1 Claims of the OMT Method

The OMT (Object Modelling Technique) method includes both analysis and design. The term '*object-oriented*' used in this method means that object-oriented software is a collection of *discrete objects that incorporate both data structure and behaviour*. In analysis, this method aims to address the themes: *abstraction, encapsulation (also information hiding), combining data and behaviour, inheritance*, and *emphasis on object structure but not procedure structure*. Underlying these themes, in order to specify a system, three kinds of model are built: *object model, dynamic model, and functional model*, describing application-domain objects rather than computer-domain objects. These three models aim to provide three cross-referenced views of a system: the object model shows the objects and their relationships in the system and it is the fundamental model; the dynamic model describes the behaviour of objects; and the functional model specifies the data transformations in the system. These models are assumed to provide a basis for an agreement between users and developers, and to make the specification of the system feasible for later design. The method states that a successful analysis model should be a concise, precise and understandable abstraction of what a system must to do rather than how it is done.

To build these models, an *iterative process* of analysis is provided by the method, based on a series of steps as follows:

- *Constructing the object model, constructing the dynamic model, constructing the functional model, adding operations, iterating the analysis.*

During analysis, analysts are required to communicate with users to avoid ambiguities and misconceptions.

2.2.2.2 Inputs Required

The inputs to analysis required by this method include the *problem statement* that describes an application scenario and gives a conceptual survey of the desired system, the dialogues with users, and the knowledge from the real world. The latter two inputs are needed when the initial problem statements are not complete or correct.

2.2.2.3 Analysis Addressed

From the claims of the OMT method above, we see that this method does analysis by an iterative process to build three models. In this section, we describe how we used this method to analyse the given book trader scenario, as in Section 2.2.1.

Step 1: Constructing an Object Model

This step includes eight substeps to build an object model. The method supposes that the order of these substeps could be interchanged or combined, if necessary.

(1) Identifying Objects and Classes

Objects and classes are identified from the problem statement, for which we used the given book trader scenario.

> Object—An *object* is a concept, abstraction, or thing with crisp boundaries and meanings for the problem. It is an instance of a class.
>
> Class—A *class* is a description of a group of objects with similar properties, common behaviour, common relationships and common semantics.

The method assumes that objects and classes often correspond to nouns in the problem statement, therefore this step firstly lists all nouns found from the scenario and then selects the significant classes according the criteria given by the method: they should not be a) redundant, b) irrelevant, c) vague, e.g., too broad in scope, d) attributes, i.e., they do not primarily describe individual objects, e) operations (i.e., they should be manipulated in their own right), and f) implementation constructs, such as a linked list. Additionally, not

all objects appear explicitly in the problem statements and some *extra* objects may be added in the model later.

Using these guidelines, the nouns were abstracted from the book trader scenario(see Figure 2.8), and then selected by the criteria above. Figure 2.9 shows the four classes that are left as the significant classes for the book trader system.

(2) Preparing a Data Dictionary

A *data dictionary* keeps a record of all modelling entities. For each class, a precise description is written, and example data dictionary entries for the significant classes of the book trader system are given in Figure 2.10.

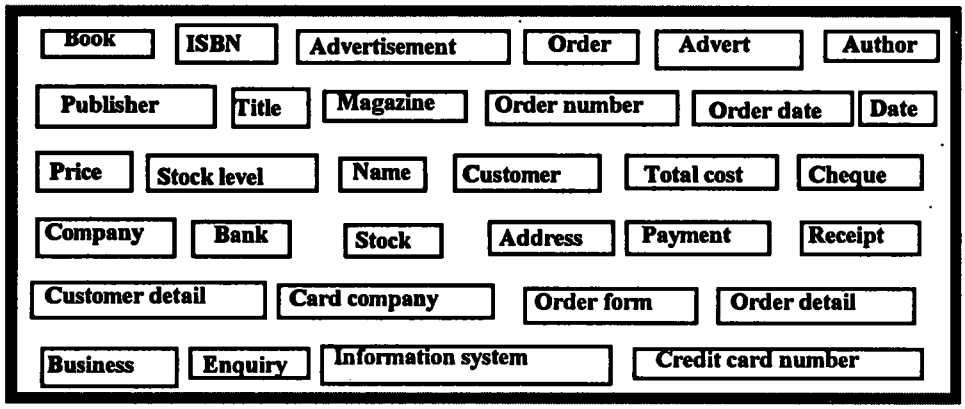

Figure 2.8 Nouns Abstracted from the Book Trader Scenario

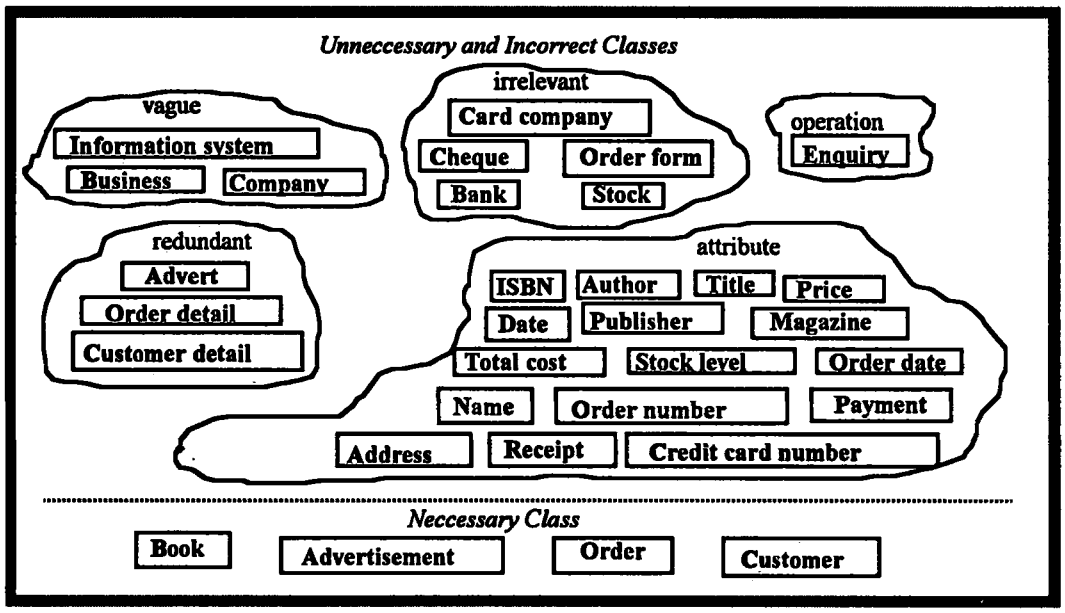

Figure 2.9 Significant Classes in the Book Trader System

Book—a book that will be advertised in magazines and ordered by customers. A book may be ordered by more than one order form. A book is described by the unique ISBN, its title, publisher and cost price. A book may have more than one selling price.

Advertisement—a description of books in a magazine on a given date: their titles, authors, publishers, selling prices. A book may have different selling prices in different advertisments. The number of the orders corresponding an advertisement needs to be recorded in order to reply to the enquiries about it.

Order—a request from a customer for buying one or more books. It includes the name and address of the customer, the title and selling price of each book, total cost of the ordered books and the method of payment, i.e., either a cheque or a credit card.

Customer—the buyer of books who is recorded with the name and the address. A customer knows about the books by reading the advertisements published. Then he/she can either post a filled order form or telephone the company to buy books.

Figure 2.10 A Data Dictionary of the Classes in the System

(3) Identifying Associations

Any dependency between two (or more) objects is represented as a link. A reference from one class to another is an association.

Link—A *link* represents a physical or conceptual connection between objects, that is, a relationship between two (or more) objects.

Association—An *association* is a group of links with common structure and semantics.

An association is inherently bi-directional although the name of an association is usually read from left-to-right in a diagram. The name of an association can be omitted if a pair of classes has a single association whose meaning is obvious. In addition, the OMT method supposes that a particular kind of association, *aggregation,* is an association with some extra semantics, unless two objects are tightly bound in a whole-part relationship. The distinction can be made by thinking about "Would you use the phrase part of? Are the operations on the whole automatically applied to its part? Are some attribute values propagated from the whole to all or some parts?".

Potential associations are first identified by abstracting the *verb phrases* from the scenario. Figure 2.11 lists the verb phrases that were abstracted from the book trader scenario.

Verb Phrases
selling books by advertising in magazines
book may have a different price in different advertisements
customer can buy one or more of the advertised books
mailing the company an order form
payment is required
telephone the company
a receipt is produced
taken from stock and sent to the customer
the bussiness is supported by an information system
requests the number of orders
requests details of the progress of a specific order
customer is only relevant to a specific order
identifying the order

Figure 2.11 Verb Phrases from the Book Trader Scenario

Associations are then selected if they are not: a) associations between eliminated classes, b) irrelevant or implementation associations, c) actions or events, d) ternary associations (that is, they need to be decomposed into binary associations), and e) derived associations (i.e., redundant associations). The multiplicity of an association also needs to be specified in a model, such as one-to-many or many-to-many.

These criteria were applied to the verb phrases in Figure 2.11, and the rejected ones were as follows:

- irrelevant or implementation association—taken from stock and sent to the customer, the business is supported by an information system, identifying the order
- action—payment is required, a receipt is produced, request the number of orders, requests details of the progress of a specific order
- derived association—customer can buy one or more of the advertised books.

The remaining verb phrases represent the useful associations between the classes, described as follows and represented by an object diagram shown in Figure 2.12:

- *Order-Book* association—An order must be for one or more books, and a book may never be ordered or may be ordered many times.
- *Order-Customer* association—A customer should be only relevant to a specific order.
- *Order-Advertisement* association—An order corresponds to only one advertisement.
- *Book-Advertisement* association—A book may never be advertised or may be advertised many times, and an advertisement must contain one or more books.

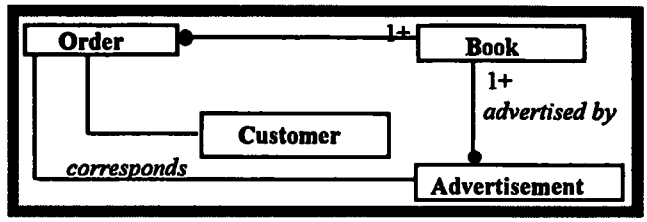

Figure 2.12 Initial Object Diagram for the Book Trader System

(4) Identifying Attributes of Classes and Links

This substep identifies the attributes of classes.

Attribute—An *attribute* is a named property of a class describing a data value held by the objects in a class.

The method's instructions are to first identify the attributes that directly relate to a particular requirement in the scenario, and to add more details later. According to the method, a significant attribute should not be: a) *an object* (i.e., it should not have its own features), b) *a qualifier* (i.e., the values of it do not depend on a particular context), c) *a name* that depends on the context, d) *an identifier*, e) *a link attribute (or object)* that depends on the presence of a link, f) *an attribute* for which the values are invisible to the outside, g) *an attribute* that is unlikely to affect most operations, and h) *a discordant attribute* that seems completely different from and unrelated to all other attributes in the same class (i.e., this class should be split into two distinct classes). In the object model, one-to-many and many-to-many associations may be qualified with a qualifier.

Using the initial guidelines, the candidate attributes of each class in the book trader system were identified and put in the object model as shown in Figure 2.13(a).

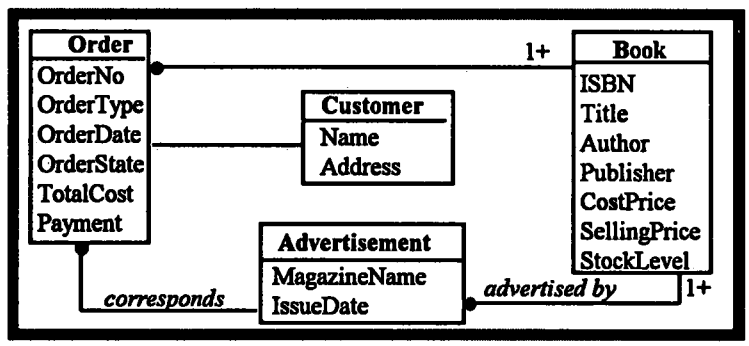

Figure 2.13(a) Attributes of the Objects and Classes in the System

Each attribute was then checked by the criteria above:

- the independent existence of attribute 'Payment' within *Order* is important because it has its own features 'payment type' (either a cheque or a credit card) and 'payment detail' (either cheque detail or credit card detail). It is thus promoted as a *part* class *Payment*. The *whole* class of it is *Order*.
- attribute 'ISBN' of *Book* is a qualifier: an advertisement plus an ISBN yields a unique book. The many-to-many association 'advertised by' is therefore changed into a many-to-one association with this qualifier.
- attribute 'SellingPrice' within *Book* is a link object since the value of it depends on the link 'advertised by'.

A revised object model is shown in Figure 2.13(b) so far.

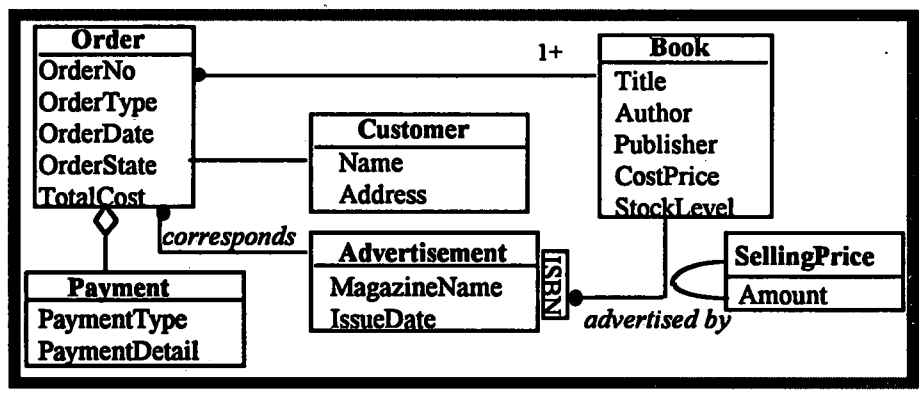

Figure 2.13(b) A Revised Object Model of the Book Trader System

(5) Identifying Inheritance

An OMT inheritance structure has generalisation and specialisation that are found by searching for classes with similar attributes, associations and operations.

Inheritance—An *inheritance* is a mechanism that permits classes to share attributes and operations based on a relationship, usually generalisation.

Generalisation—A *generalisation* is the relationship between a class and one or more refined versions of it.

Specialisation—A *specialisation* is the creation of subclasses from a superclass by refining the superclass.

Superclass—A *superclass* is the class that is refined.

Subclass—A *subclass* is the refined version of a superclass.

Multiple inheritance possibly increases sharing but it also increases the complexity of both concepts and implementation. Attributes and associations should be assigned to the most

general class for which it is appropriate. No inheritance structure was built in the book trader system by this substep.

(6) Testing Access Paths

In order to check that the required results can be obtained, access paths through the object model should be traced. The object model in Figure 2.13(b) was checked that the associations provided appropriate access paths.

(7) Iterating Object Modelling

This substep checks the overall consistency of an object model. This includes identifying possible missing objects and associations, deleting unnecessary classes and associations, and checking if there is incorrect placement of associations or attributes. The object model in Figure 2.13(b) was checked and it was found that classes *Order* and *Payment* play two roles respectively: mail and phone orders and cheque and credit card payments. The class *Payment* is thus replaced by new classes *ChequeOrder* and *CreditCardOrder* that are the subclasses of *Order* so that each class plays one role in the object model, as shown in Figure 2.13(c).

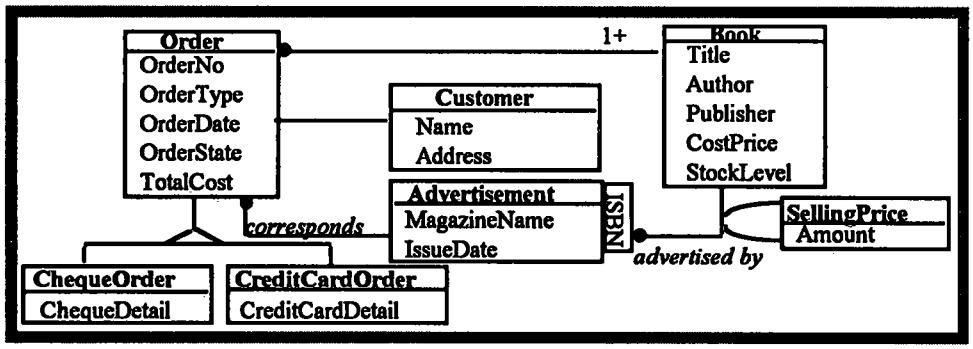

Figure 2.13(c) A Further Revised Object Model of the System

(8) Grouping Classes into Modules

For a large problem, it may be necessary to divide the object model diagram into sheets of uniform size for convenience in drawing, printing and viewing. To aid this, tightly-coupled classes should be grouped together in a module.

Module—A *module* is a set of classes (one or more sheets) that captures some logical subset of the entire model.

The object model of the book trader system is drawn on one sheet and so there is only one module in this system.

Step 2: Constructing a Dynamic Model

The dynamic model is represented by state diagrams that show the behaviour of the objects of each class by representing the events relating to it.

(1) Preparing a Scenario of Typical Interaction Sequences

This substep looks for events—*externally-visible* stimuli and responses. A scenario is analysed here since it shows the major interactions, external display formats and information exchanges.

Scenario—A *scenario* is a sequence of events.

Event—An event is a signal, input, decision, interrupt, transaction, or action.

The normal cases are first identified for a normal scenario, and then the special cases, such as maximum or minimum values and error cases, are analysed as a scenario of exceptions. Other cases may be added in these two kinds of scenario. A normal scenario developed for the book trader is listed in Figure 2.14, and a special scenario is given in Figure 2.15 .

Advertisements tell the details of the books in stock.
A customer buys the advertised books by mailing an order.
A customer buys the advertised books by phoning an order.
The detail of the order is recorded.
The detail of order is checked.
The payment is valid.
The order produces a receipt.

Figure 2.14 A Normal Scenario of the Book Trader

Advertisements tell the details of the books in stock.
A customer buys the advertised books by mailing an order.
A customer buys the advertised books by phoning an order.
The detail of the order is recorded.
The details of the order are found incorrect, or the payment is invalid.
This order is rejected.

Figure 2.15 A Special Scenario with Exceptions

(2) Identifying Events between Objects

From the scenarios of the previous step, normal events, error conditions and unusual events are identified. An event trace list is used here to show an ordered list of

events between objects. An event flow diagram is then drawn to show the events between objects of classes. Figure 2.16 and Figure 2.17 show, respectively, an event trace list and an event flow diagram developed for the scenario of the book trader.

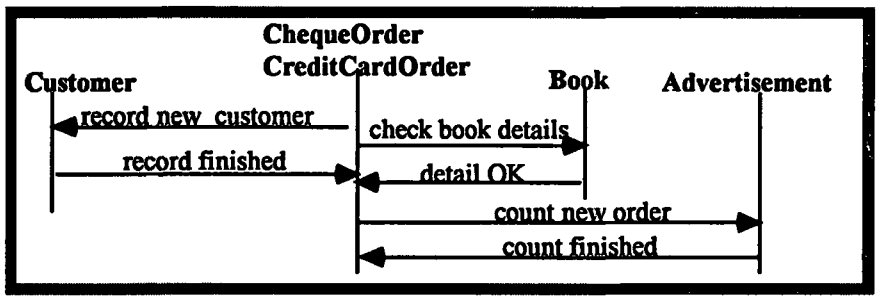

Figure 2.16 An Event Trace List for the Normal Scenario

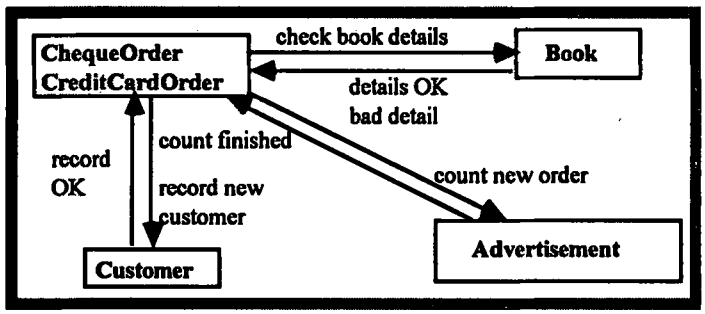

Figure 2.17 An Event Flow Diagram for the Book Trader System

(3) Drawing a State Diagram for Objects of Each Class

OMT uses a state diagram to specify the behaviour of the objects of a class. Every event trace corresponds to a path in a state diagram. The interval between any two events is a state of an object. From the event trace list above, a trace is first picked showing a typical interaction for an object. The event is then connected with a path in the state diagram, to show the sequence of events and states in an object class. If the objects of a class does not have any significant state transition there is no need for a state diagram.

For the book trader system, a state diagram was produced for the class *ChequeOrder*, and is given in Figure 2.18 . The 'initial' state is entered when an order is created, and an object is be deleted when one of the final states shown by a bull's eye is entered. The actions such as 'produce a receipt' follow the sign 'do:'.

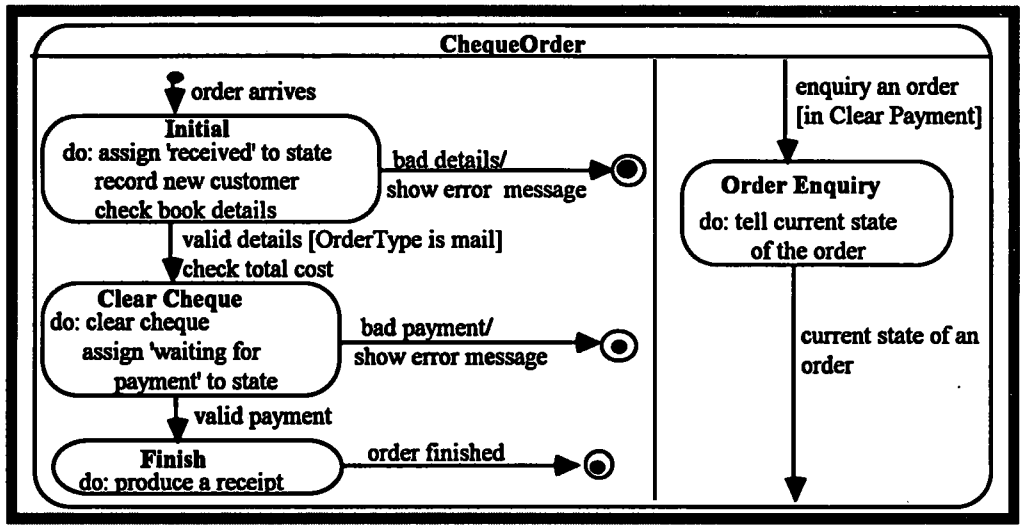

Figure 2.18 A State Diagram for the Class *ChequeOrder*

(4) Match Events between Objects to Verify Consistency

State diagrams must satisfy the criteria: a) every event should have a sender and a receiver; b) states without predecessors or successors should represent starting or terminating points of the interaction sequence; c) events through the system should match the scenarios; d) corresponding events on different state diagrams should be consistent; and e) synchronisation errors where an input occurs at an awkward time should be prevented since objects are inherently concurrent. The state diagram shown in Figure 2.18 satisfies these criteria.

Step 3: Constructing a Functional Model

A functional model shows the transitions of values from inputs to outputs. It is represented by data flow diagrams that illustrate the functional dependencies in a system. The method supposes that the processes in a data flow diagram correspond to the activities or actions in a state diagram, and the data flows in it correspond to the attribute values for objects defined in an object diagram.

(1) Identifying Input and Output Values

Input values from outside and *output values* from inside system should be identified first from the scenario to decide the *boundary* of the system. Input values are the parameters of input events that affect the control flow in a system. Output values are the

results from the system. Figure 2.19 shows the input and output values of the book trader system.

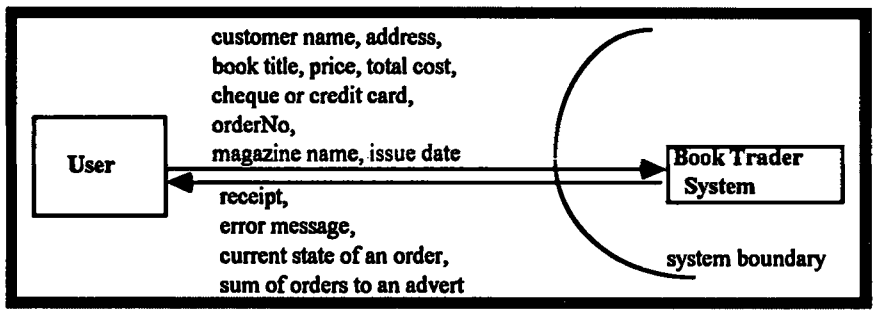

Figure 2.19 Input and Output Values of the Book Trader System

(2) Building Data Flow Diagrams Showing Functional Dependencies

A *data flow diagram* shows how input values are transformed into output values by a *process* without sequencing the functions. Data flow analysis is a process of functional decomposition: a top data flow diagram is drawn first, then a complex process within it is decomposed to produce a lower level diagram. This analysis continues until every process in the diagram performs a simple function. Figure 2.20 shows a data flow diagram developed for the book trader system, in which 'user' is drawn as a terminator and the classes *ChequeOrder* and *Advertisement* are the *actors* that produce or consume values. The classes *Book*, *Customer*, and *SellingPrice* are *data stores* that store values for access during processing.

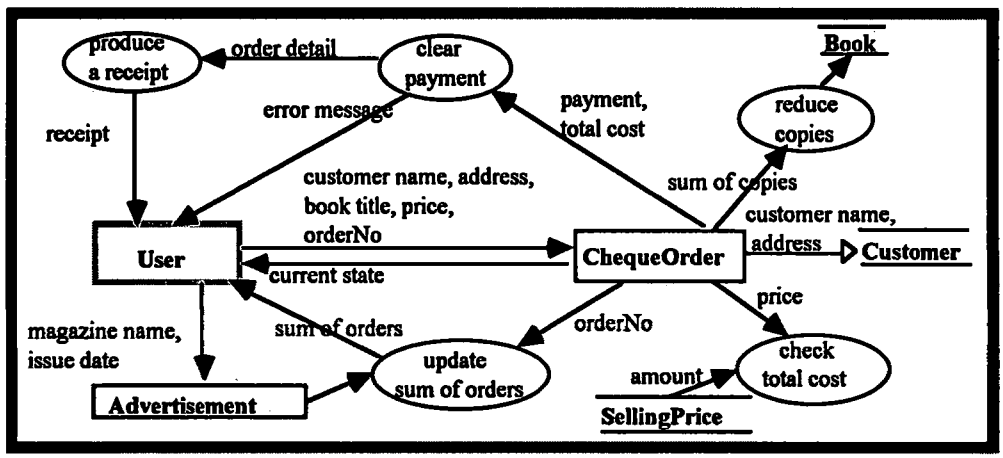

Figure 2.20 A Data Flow Diagram of the Book Trader System

(3) Describing Functions

Each function can be specified in a declarative or procedural form. A declarative specification shows the relationships between input and output values or between the output values; while a procedural specification gives an algorithm for the function. This description specifies what the function does but not how it is implemented. Figure 2.21 shows a description of the function *clear payment.*

```
clear payment (payment, total cost) ->order detail, error message
        If payment is accepted
                send order detail to the process 'produce receipt'
          else  print 'invalid payment'
```

Figure 2.21 A Description of the Function *clear payment*

(4) Identifying Constraints between Objects

Constraints between objects are functional dependencies but not input-output dependencies. They are specified as pre/postconditions on operations. A constraint in the book trader system, for example, is 'the value of stock level of book is never negative'.

(5) Specifying Optimisation Criteria

This step is concerned with the implementation of functions rather than their specification, and so we do not consider it in this study.

Step 4: Adding Operations

This step adds operations to the classes in an object model.

Operation—An *operation* is a function or transformation that may be applied to or by objects in a class.

Operations can be abstracted from the attributes and associations in an object model, or from the events in a dynamic model, or the processes in a functional model.

(1) Operations from the Object Model

The operations that change attribute values or associations are implied by the existence of attributes and associations in the object model. These operations are thus not explicitly defined in the object model.

(2) Operations from Events

Each event relating to a class may imply an operation.

(3) Operations from State Actions and Activities

Associated with events in a state diagram are actions and activities, and if they have significant computational structure they should be considered as operations for classes. Some operations were thus added to the object model of the book trader system, as follows:

Order—check total cost, check book detail, record new customer, produce receipt, tell current state,

ChequeOrder—check order type, clear cheque,

CreditCardOrder—clear credit card.

(4) Operations from Functions

Each function in a functional model should be an operation for some class in the object model. Extra operations were therefore added to the classes *Order, Book* and *Advertisement* as follows:

Order—update order state,

Book—update stock level,

Advertisement—update sum of orders, tell sum of orders.

(5) Defining Shopping List Operations

Shopping list operations are the operations that are not dependent on a particular application; that is, they are defined for potential development in the future. This substep is ignored here since the study does not take this into account.

(6) Simplifying Operations

The operations on the object model may be simplified by a) introducing new superclasses if needed to reduce the number of distinct operations, and b) locating operations at the correct levels with the class hierarchy. On examination of the specified

operations for the book trader system using these guidelines, no operations were simplified.

The operations for a class are put in the bottom part of the box representing the class in an object model. The model for the book trader system is shown in Figure 2.22

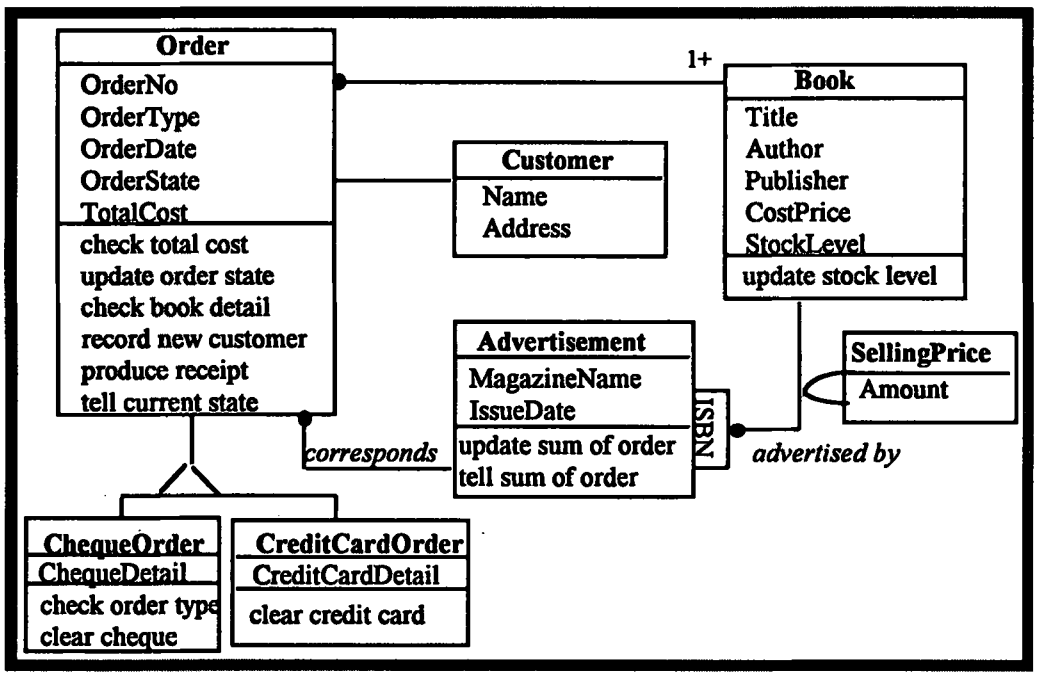

Figure 2.22 The Object Model with Operations for the System

Step 5: Iterating the Analysis

All three models are checked in this step, and confirmed by the requester and application domain experts. The models may be modified or refined by iterating the process of analysis, if necessary.

In checking the completeness and consistency of the three models for the book trader system built above, it was found that the object model is inconsistent with the behavioural model for 'sum of orders'. The object model allows the operation 'sum of orders' in the object class *Advertisement* to be calculated at any time, assuming the objects of the classes *ChequeOrder* and *CreditCardOrder* are persistent, while the behavioural model performs the 'sum of orders' instantly since objects of the classes *ChequeOrder* or *CreditCardOrder* are assumed to be deleted at some stage. A new attribute 'SumOfOrder'

is added in the class *Advertisement* in this case, in order to eliminate this inconsistency, as shown in Figure 2.23.

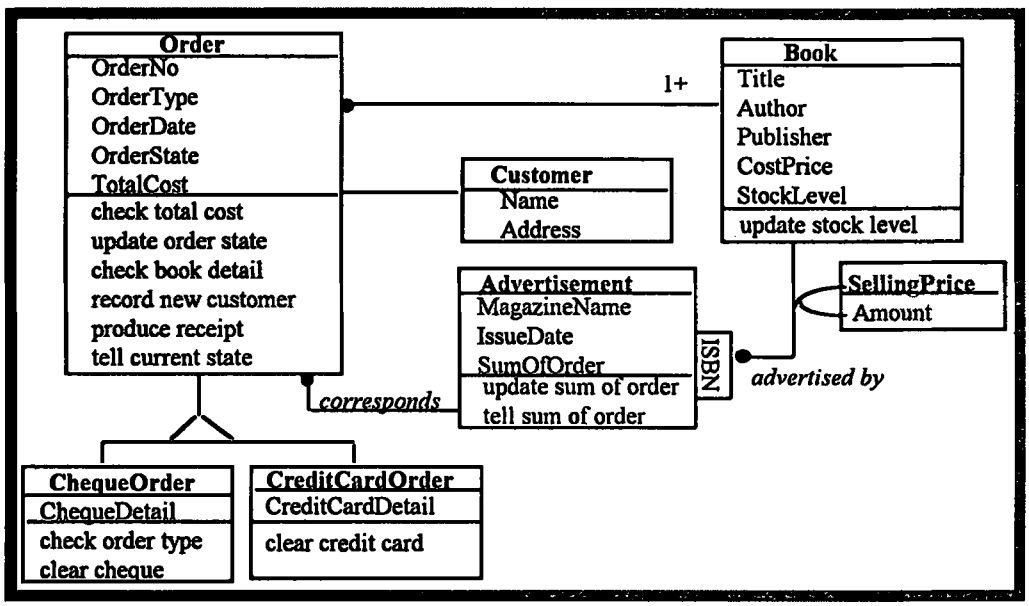

Figure 2.23 The Revised Version of the Object Model

2.2.2.4 Products Generated

The products generated from the analysis include the object, behavioural and functional models of the book trader system, expressed by (a) a data dictionary, (b) an object diagram, (c) state diagrams, and (d) a data flow diagram.

2.2.3 The Study of the Booch Method

2.2.3.1 Claims of the Booch Method

The Booch method claims to provide a path through requirement analysis to system design [Booch91]. The term '*object-oriented*' means, in this method, *objects, classes, inheritance* plus *aggregation* since the author thinks they are fundamental in object-oriented languages. The principles of *abstraction, encapsulation (also information hiding), modularity*, and *hierarchy*, are used by this method. Based on these principles, the Booch method builds an object model that describes what a system does. To build such an object model, the Booch method provides a '*round-trip gestalt*' process, i.e.,

analyse a little, design a little, for identifying objects and classes from a problem domain. This process includes four steps:

- *identifying classes and objects, identifying the semantics of classes and objects, identifying the relationships among classes and objects, and implementing classes and objects.*

2.2.3.2 Inputs Required

The inputs of analysis required by the Booch method is a problem domain when starting analysis or an incomplete model during analysis.

2.2.3.3 Analysis Addressed

From the claims of the Booch method given above, we know that this method promises to do analysis and design by providing a '*round-trip gestalt*' process: *analyse a little, design a little*. An object model is built through this process. In this section, we use the Booch method to analyse the book trader scenario, as we did for the OOA method and the OMT method.

Step 1: Identifying Classes and Objects

This step identifies candidate classes and objects from a problem domain.

Object—An *object* has *state, behaviour* and *identity* (synonym: an Instance).

Class—A *class* is a set of objects that share the structure and behaviour.

Property—An inherent or distinctive characteristic, trait, quality, or feature that contributes to making an object unique. All properties have some values. Such a value might be a simple quantity, or it might denote *another* object.

State—A state encompasses all of the (usually static) properties of the object plus the current (usually dynamic) values of each of these properties.

Field—A repository for part of the state of an object; collectively, the fields of an object constitute its structure.

Identity—An *identity* is the nature of an object that distinguishes it from all other objects.

The Booch method identifies the candidate classes and objects by focusing on the tangible things, the roles and the events that are assumed essential to the problem domain. The method does not give a specific strategy and skill to do such identification. Instead it

suggests using other existing analysis methods to do that, such as structured analysis methods (e.g., [Ward85]) or object-oriented analysis methods (e.g., [Shlaer88] and [Coad91a]). Problem domain experts may be involved in this step to help understand the vocabulary in the scenario. Therefore, for the book trader scenario, we took the same five initial classes that were identified by using the OOA method in Section 2.2.1 as the candidate classes and objects to be used for this method:

- Things remembered—*Book, MailOrder, PhoneOrder, Advertisement*
- Role—*Customer.*

Step 2: Identifying the Semantics of the Classes and Objects

This step describes the semantics of the classes and objects identified in step 1.

Behaviour—*Behaviour* is how an object acts and reacts, in terms of its state changes and message passing.

Method—A *method* is an operation upon an object, defined as part of the declaration of a class; all methods are operations, but not all operations are methods because some operations may be expressed as free subprograms.

Operation—An *operation* is an action that one object performs upon another in order to elicit a reaction (synonym: a message).

To identify the semantics of these candidate classes and objects, the method provides a strategy: first, write a script for each class and object, in order to identify the functional semantics (i.e., operations); then define the semantics of its interface; and finally define the *time and space semantics* of the object (i.e., the behaviour of the object) by specifying the sequence of its operations and states. On the one hand, this may involve going back to the previous step to shift the meanings or boundaries of existing candidate classes and objects. On the other hand, new classes and objects may be defined in this step.

(1) Write a Script for Each Class and Object

The script for candidate classes and objects can be produced by focusing on the information that needs to be stored and transformed in a system. The script produced for each class and object of the book trader scenario is shown as follows:

a) *Customer* needs to keep track of the name and address of a customer:

- Customer name, Customer address

b) *Book* needs to keep track of the following information for a book:

- ISBN, Title, Author, Publisher, Cost price, Selling price, Stock level

It is advertised for sale by the company and it can be bought by customers.

c) *Advertisement* needs to keep track of the following information for advertisements:

- Magazine name, Issue date

It advertises the books sold by the company.

d) *Mail Order* or *Phone Order* needs to keep track of a distinguishing order number, the date of order and payment detail such as a credit card number:

- OrderNo, Current state, Payment Detail

However, different constraints on payment are given for a mail order and a phone order: for a mail order, the ordered book(s) might be paid by a check or a credit card; but for a phone order, it is only allowed to pay by a credit card. The different kinds of information would be kept for them separately as follows:

—in a mail order, the following information is required:

- Payment method, Payment Detail

—in a phone order, the following information is required:

- Credit Card Detail.

(2) Defining Operations For Each Class and Object

From the above scripts, the operations for each class and object in the book trader system can be derived by considering the processing of the information it requires. The Booch method, however, does not give guidelines and criteria for the definition of the operations. We defined the operations for the classes and objects in the book trader system as follows, in order to reflect the semantics of these classes and objects:

a) Customer

- Access customer information

b) Mail Order

- Check order information, Calculate total cost, Clear payment

c) Phone Order

- Check order information, Calculate total cost, Clear credit card payment

d) Book

- Access book information, Update selling price, Update stock level

e) Advertisement

- Calculate total number of orders, Access advertisement information.

Step 3: Identifying the Relationships among the Classes and Objects

This step identifies the relationships among the classes or the objects. New classes and objects may be identified or invented, depending on the definitions of the relationships. Four kinds of relationships may exist between classes:

Using relationship—A *using relationship* is a relationship that refers to the outside view of an abstraction: a class can use another class.

Inheritance relationship—An *inheritance relationship* is a relationship among classes in which one class shares the structure or behaviour of other class(es), that is, a subclass inherits the structure or behaviour of its superclass(es).

Instantiation relationship—An *instantiation relationship* implies a process of filling in the template of a generic class to produce a class from which one can create instances.

Metaclass relationship—A *metaclass relationship* is a relationship between a metaclass and other classes (where a *metaclass* is a class whose instances are classes).

One kind of relationships may exist between objects:

Use Relationship—A *use relationship* implies the ability to send messages along the path between two objects.

(1) Drawing Class Diagrams

A class diagram for the Booch method represents the classes and their relationships in a system. The following diagrams show how the relationships among the classes for the book trader system were defined. Note that, because some operations are the same in classes *Mail order* and *Phone order*, we created a superclass *Order* that includes the common operations, using Booch notation as follows:

Name: Order
Documentation: The order from a customer for buying one or more books
Cardinality: 1/1/n
Hierarchy:
Superclasses: None
Metaclass: None
Operations: Check order information, Calculate total cost.

Two *inheritance* relationships were thus defined among these classes, as in Figure 2.24 below. Three *use* relationships were defined between the class *Order* and the classes *Advertisement (1:n), Customer(1:1), Book (n:n)*, as also illustrated in Figure 2.24.

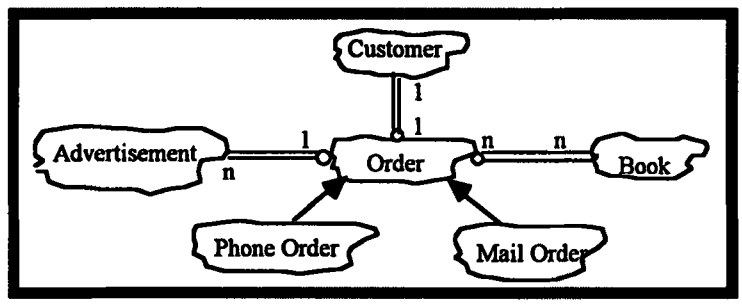

Figure 2.24 The Class Diagram of a Book Trader System

To describe the behaviour of a class, a state transition diagram is used by the Booch method to specify the state changes resulting from events and the actions causing these changes. For the class *Order*, its state is different before and after clearing a payment; the state changes for this class are described by a class diagram shown in Figure 2.25.

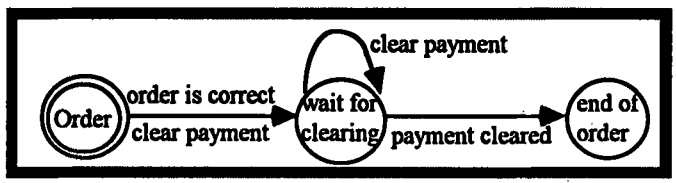

Figure 2.25 The State Transition Diagram for the Class *Order*

(2) Drawing Object Diagrams

An object diagram shows the objects and the relationships between objects. An object *B* may be visible to another object *A* in the following forms:

- *Same lexical scope*: *B* is within the scope of *A*; thus *A* can explicitly name *B*.
- *Parameter*: *B* is passed as a parameter to some operation applicable to *A*.
- *Field*: *B* is a field of *A*.

The Booch method requires that the operations appearing in an object diagram must be consistent with the operations defined in the associated classes.

To draw the object diagram of the book trader system, three message connections between the object *anOrder* and the other objects *aBook*, *aCustomer* and *anAdvertisement* were identified, as drawn in Figure 2.26, as well as three field visibilities in which the object *anOrder* is visible to the other objects.

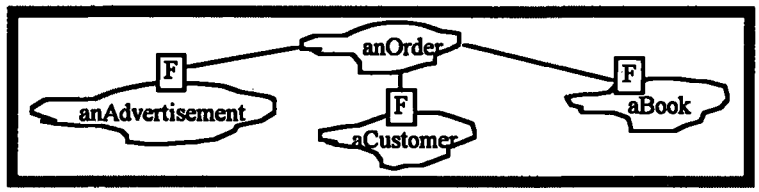

Figure 2.26 The Object Diagram of the Book Trader System

In addition to the class and object diagrams for a system, the detail of a class is given by filling in a template. For example, a more complete template for the class *Order* is given as follows:

Name: Order
Documentation: The order from a customer for buying one or more books
Visibility: Private
Cardinality: 1/1/n
Hierarchy:
    Superclasses: None
    Metaclass: None
Implementation:
    Fields: Current state, theCustomer, theBook, theAdvertisement
    Operations: Check order information
        Calculate total cost
State transition diagram: Figure 2.25.

Step 4: Implementing the Classes and Objects

This step allocates the classes and objects defined in the above steps into the modules that describe how these objects and classes are implemented by object-oriented design. We do not take this into account in this study.

2.2.3.4 Products Generated

The analysis supported by the Booch method produces (a) an object diagram, (b) a class diagram, and (c) a state transition diagram.

2.2.4 The Study of the Wirfs-Brock Method (Wirfs-Brock et al.)

2.2.4.1 Claims of the Method

The authors of the Wirfs-Brock method [Wirfs90] claim that this method addresses the issues, *abstraction, encapsulation, information hiding, inheritance* and *message-sending*, in analysing and designing object-oriented systems. Underlying these issues, it views the real world as a system of operating and collaborating computational objects. A scenario from the real world is thus a system in which there are many objects and an object may send a request to another object in order to perform an operation (i.e., responsibility) or to reveal some of its information, or both. This method also emphasises the functional collaborations among classes in a system: the responsibilities of a class, the client-server relationships and the contracts among the classes in the system.

To find classes and their collaborations from a system specification, this method provides a process based upon the responsibility-driven tactic of analysis that includes an *exploratory* phase and an *analysis* phase. The exploratory phase includes the steps:

- *Finding classes, finding responsibilities of classes, finding collaborations among classes.*

The analysis phase includes the steps:

- *identifying hierarchies for classes, identifying subsystems, and constructing the protocols for each class.*

A set of graphs and cards are produced by these two phases.

2.2.4.2 Inputs Required

This method assumes that the input of analysis is a specification of system requirements that reflects the real world and that is written in a natural language.

2.2.4.3 Analysis Addressed

The claims of the Wirfs-Brock method show that this method addresses the analysis of object-oriented systems through two phases. To find how this method does analysis, we

took the book trader scenario as a specification of a system and used this method to model a book trader system, as for the other three methods.

Step 1: Finding Classes

This step abstracts classes from a system using the vocabulary of the system specification.

Object—An *object* encapsulates both functions and data. That is, it retains information, and knows how to perform certain operations.

Class—A *class* is a generic specification for an arbitrary number of similar objects which share the same behaviour. Objects in a class are called instances of that class.

(1) Looking for Noun Phrases

All noun phases were first extracted from the book trader scenario, as listed in Figure 2.27.

customer	company	credit card number
selling book	unique ISBN for each book	credit card
advertised book	order details	cheque
cost price	price of book	order number
order form	advert	order date
advertisement	name of customer	receipt
magazine	address of customer	way of paying
payment	method of payment	authorization
author of book	total amount	ordered book
publisher of book	title of book	stock level of book
order	total cost	customer detail
information system	order detail	bank
name of magazine	card company	stock
date of issue	business activity	enquiry
floor limit		

Figure 2.27 The Noun Phrases in the Book Trader Scenario

(2) Choosing Meaningful Candidate Classes

Only the meaningful classes are put into a system. They are determined by the criteria: a) keep physical objects, b) keep conceptual entities, c) one word for one concept, d) if the meaning of adjectives for the same noun implies different objects, they are defined as different classes, e) if the missing subject for a sentence is a potential object, a new class may be defined, f) if a noun phrase seems outside the system, do not include it as a class, and g) values of attributes may be defined as classes.

For the book trader system, the meaningful classes from the list in Figure 2.27 were selected by first discarding the obviously irrelevant ones, as shown in Figure 2.28.

Not Releavant		
bank	card company	company
stock	information system	cheque
floor limit	business activity	authorization
order form	credit card	magazine
Possible Meaningful Classes		
customer	order	credit card number
selling book	unique ISBN for each book	order number
advertised book	price of book	order date
advertisement	advert	receipt
payment	name of customer	method of payment
author of book	address of customer	stock level of book
publisher of book	title of book	enquiry
name of magazine	total cost	way of paying
date of issue	total amount	ordered book
cost price	order detail	customer detail
different price		

Figure 2.28 Two Categories of Noun Phrases

Candidate classes are then identified according to the criteria above:

a) Physical objects

— 'selling book', 'advertised book', 'ordered book', 'advert' and 'advertisement'.

b) Conceptual entities

— 'customer', 'order', and 'enquiry'.

c) One word for one concept

— 'book' instead of 'selling book', 'advertised book', and 'ordered book';

— 'advertisement' instead of 'advert';

— 'customer detail' and 'order detail' are overlapped by other noun phrases such as 'name of customer' and 'order number';

— 'total cost' instead of 'total amount';

— 'cost price' instead of 'price of book';

— 'method of payment' instead of 'way of paying'.

d) Different adjectives for different classes

—The 'cost price' means the original cost of a book.

—The 'different price' means the selling price of a book, i.e., a book may have different selling prices. Therefore, it is more appropriate to use the phrase 'selling price' instead of 'different price' to name the class.

e) Missing subjects that may not appear in the specification explicitly but they may have to be defined as new classes because of the needs of the system

—For the sentence 'each advertisement is placed in one magazine on a given date', the system is not concerned with who is responsible for placing an advertisement in a magazine. So no new class is needed here.

—For the sentence 'Payment is required', the missing subject is 'Order'.

—For the sentence 'Once payment for an order is approved, a receipt is produced, the ordered books are taken from the stock and sent to the customer with the receipt', 'Order' can be considered as the missing subject.

—For the sentence 'the unique ISBN for each book is recorded, together with its title, the names of its authors, the name of its publisher, its cost price and its current stock level (i.e., the number of the copies of the book in the company's warehouse)', 'Book' can be considered as the missing subject.

—For the sentence 'for each advertisement, the name of the magazine in which it appears and the date of the issue is recorded, as well as the list of books and their price given in the advertisement, 'Advertisement' can be considered as the missing subject.

—For 'Each order is assigned a distinguishing order number and details are recorded of the customer, the date, the books required, the total cost and a credit card number if that is the chosen method of payment', 'Order' and 'Customer' can be considered as the missing subjects.

Thus no more new candidate classes are needed.

f) Unnecessary classes

—As required by the requirements, the 'total cost' and 'cost price' are attributes of the class 'Order'.

g) Attributes and values

The attributes for each meaningful candidate class are listed as follows:

Book—ISBN, title, author, publisher, cost price, stock level

Advertisement—name of magazine, date of issue

Customer—name, address

Order—order number, type, date, total cost, payment, receipt

Enquiry—time, type

Selling Price—amount.

No value of these attributes would be defined as a class.

At this point, six candidate classes were selected for the book trader system: *Book, Advertisement, Customer, Order, Enquiry* and *Selling Price*.

(3) Finding Missing Classes

Some missing classes (i.e., superclasses and abstraction classes) are identified in this substep.

Superclass—A *superclass* is a class from which specific behaviour is inherited.

Abstract class—An abstract class is a superclass that is not intended to produce instances of itself. It specifies common behaviour for a variety of classes and then these classes can inherit the common behaviour.

Subclass—A *subclass* is a class that inherits behaviour from another class, i.e., abstract class or its superclass. A class might have several abstract classes or superclasses. A subclass usually has its own behaviour as well as the inherited behaviour.

An abstract class or superclass is identified by grouping the classes that share their behaviour. For the book trader system, the above candidate classes do not share common behaviour and so no abstract or superclass was defined.

Each class is now recorded by a class card, as listed for the book trader system in Figure 2.29.

Class:	Book
Superclasses:	none
Subclasses:	none
Hierarchy Graphs:	none
Collaborations Graphs:	
Description:	This class represents the books that are advertised in magazines and ordered by customers.

Class:	Customer
Superclasses:	none
Subclasses:	none
Hierarchy Graphs:	none
Collaborations Graphs:	
Description:	This class represents the buyers who order the advertised books.

Class:	Advertisement
Superclasses:	none
Subclasses:	none
Hierarchy Graphs:	none
Collaborations Graphs:	
Description:	This class represents the advertisements that show the sale details of books.

Class:	Order
Superclasses:	none
Subclasses:	none
Hierarchy Graphs:	none
Collaboration Graphs:	
Description:	This class represents the orders of books that are provided by customers.

Class:	Enquiry
Superclasses:	none
Subclasses:	none
Hierarchy Graphs:	none
Collaborations Graphs:	
Description:	This class represents the enquiries about the number of the orders responding to an advertisement or about the progress of a specific order.

Class:	Selling price
Superclasses:	none
Subclasses:	none
Hierarchy Graphs:	none
Collaborations Graphs:	
Description:	This class represents the selling prices that are published for advertised books.

Figure 2.29 Initial Class Cards in the Book Trader System

Step 2: Defining Responsibilities

This step defines the responsibilities of each class through three substeps.

Responsibility—A *responsibility* is the knowledge an object maintains, and the actions an object can perform.

(1) Finding Responsibilities from the Purpose of Each Class

The role of a class in a system is supposed to imply its responsibilities in the system. The following shows some responsibilities identified for the classes in the book trader system according to their roles in the system:

Book—Record and access book information
Customer—Record customer information
Advertisement—Access advertisement information
Order—Record order information
Enquiry—Answer an enquiry
Selling price—Access the amount.

(2) Extracting Responsibilities from the Specification

The specification of a system includes the actions that must be performed. The actions can be abstracted from the verb phrases in the specification. Figure 2.30 lists the verb phrases taken from the specification of the book trader system.

selling books by advertising them in magazines
book may have a different price in different advertisements
customer can buy one or more advertised books
mailing an order form to the company
payment is required
telephone the company
a receipt is produced
taken from stock and sent to the customer
the bussiness is supported by an information system
requests the number of orders
requests details of the progress of a specific order
customer is only relevant to a specific order
identifying the order

Figure 2.30 The Verb Phrases from the Book Trader Scenario

Each action can be assigned as a responsibility of the class to which it logically belongs by the following guidelines: a) decide how much a class knows or can do for a system, and how many objects it can affect, b) assign the common behaviour to the superclass at the highest level, c) keep behaviour with related information, d) keep information about one thing in one place, and e) split shared responsibilities among related objects into several smaller and more specific responsibilities and assign them separately to the most appropriate classes.

The responsibilities for the book trader system were identified from the verb phrases listed in Figure 2.30 and assigned to classes by following the given guidelines:

Book—Record book information, Access book information, Update stock level
Customer—Record customer information
Advertisement—Record advertisement information, Access advertisement information, Calculate sum of response orders
Order—Record order information, Access order information, Update order state, Access order state, Clear payment, Produce receipt
Enquiry—Accept enquiry, Answer enquiry
Selling price—Access the amount.

(3) Identifying Responsibilities from the Relationships between Classes

Some responsibilities may be implied in the relationships, 'is-kind-of', 'is-analogous-to' and 'is-part-of', between classes. No more responsibilities of the classes in the book trader system were identified at this step since there are no such relationships among the classes.

Step 3: Defining Collaborations

A class may process a responsibility itself or may require another class to do the processing for it. This step is concerned with the latter co-operation between classes:

Collaboration—A *collaboration* is a request from a client to a server in order to perform a client responsibility, that is, it implies a *contract* between a client and a server.
Contract—A *contract* is a list of requests, i.e., services, that a client can make of a server.
Service—A *service* is a responsibility of a server for a contract within a class that can be requested by other objects.

To find collaborations and contacts between classes, the responsibilities with dependencies need to be identified by analysing the interactions between the classes.

(1) Identifying Collaborations by Responsibilities

Examine each responsibility by considering 'Is the class capable to fulfilling this responsibility itself? If not, what does it need? From what other class can it acquire what it needs?'. A shared responsibility often implies a collaboration between the classes.

(2) Identifying Collaboration by Classes

For each class, consider the questions 'What does this class do or know? What other classes need the result or information from this class?'. If no interaction exists between a class and any other class, this class should be discarded.

(3) Identifying Collaborations by Relationships

In addition, examine three specific relationships, 'is-part-of' and 'has-knowledge-about' and 'depends-upon', to identify possible collaborations, since these relationships often imply the interactions among classes.

The following collaborations between classes in the book trader system were thus identified by this substep:

—Order (client) with Customer, Book and Advertisement (servers),

—Enquiry (client) with Order and Advertisement (servers), and

—Advertisement (client) with Order and Selling price (servers).

Step 4: Defining Hierarchies

The hierarchy of classes is defined in this step.

(1) Building Good Hierarchies

A class hierarchy and a contract of shared responsibilities are defined by the guidelines: a) model a kind-of hierarchy, b) put common responsibilities as high as possible, and c) make sure that abstract classes do not inherit from concrete classes.

No hierarchy was built for the book trader system by this substep.

(2) Identifying Contracts

The contracts between classes and services in the classes are defined here.

Private Responsibility—A *private responsibility* is the behaviour that a class must have, but cannot be requested by other objects.

The guidelines given to define contracts are: a) group responsibilities used by the same clients, i.e., they belong to one contract, b) maximise the cohesiveness of classes, and c) minimise the number of contracts since the fewer contracts that exist the more comprehensible a system.

Under these guidelines, the contracts between the classes in the book trader system were defined as listed in Table 2.1.

contract number	contract	server	client
1.	Access order state	*Order*	*Enquiry*
2.	Record customer information	*Customer*	*Order*
3.	Access book information	*Book*	*Order*
4.	Update stock level	*Book*	*Order*
5.	Access order information	*Order*	*Advertisement*
6.	Calculate sum of response orders	*Advertisement*	*Enquiry*
7.	Access the amount	*Selling price*	*Advertisement*

Table 2.1 A List of Contacts between the Classes in the Book Trader System

Step 5: Defining Subsystems and Protocols

If a system is large, in this step it can be partitioned into a set of smaller subsystems to simplify the interactions between classes and to make the system understandable.

Subsystem—A *subsystem* is a group of classes or a group of classes and other subsystems that collaborate to support a set of contracts. The classes in a subsystem work closely together to provide a clear unit of functionality.

Protocol—A *protocol* is a set of signatures to which a class respond to.

Signature—A *signature* is the name of a method, the types of its parameters, and the type of the object which the method returns.

(1) Defining Subsystems

Subsystems are defined by decomposing an application domain into smaller sub-domains. In particular subsystems can be identified by first drawing a collaboration graph of the system that shows a web of the many collaborations between classes, and then considering the questions "What is the purpose of that web? Does it show these classes work together to implement a unit of functionality? Does it make sense to abstract the group of classes out as a single entity? Can a subsystem be built in order to subsume these classes?". One way to determine whether a group of classes is a subsystem is to name it. If it can be named, it is likely a subsystem.

Using these guidelines, the collaboration graph of the book trader system was drawn as in Figure 2.31. No subsystem was defined since the interactions between the classes in the system are not complex.

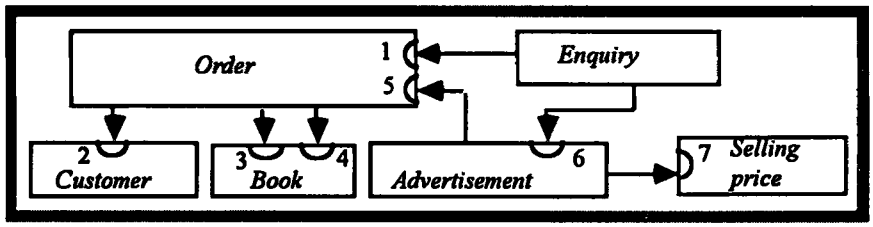

Figure 2.31 The Collaborations Graph of the Book Trader System

(2) Constructing Protocols for Each Class

From a collaboration graph, the protocols of each class can be defined based upon the contracts in this graph, using the following guidelines: a) use a single name for each conceptual operation, b) associate a single conceptual operation with each method name, and c) if classes fulfil the same specific responsibility, define it explicitly in an inheritance hierarchy.

The protocols of the classes in the book trader system were defined as listed in Table 2.2.

contracts	protocols
Access order state	orderState() returns current state
Record customer information	recordCustomer(name, address) returns OK
Access book information	sendBookDetail() returns book information
Update stock level	update(stock level) returns new value of stock level
Calculate sum of response orders	sumOrder() returns number of orders
Access order information	sendOrderDetail() returns order information
Access the amount	amountPrice() returns current value of amount

Table 2.2 The Protocols of the Classes in the Book Trader System

(3)Writing a Design Specification for Each Class and Subsystem

The specification for each class and subsystem is recorded on a class card or a subsystem card, based on the guidelines: a) redraw the graphs on pages, one page per graph, b) number the pages so they can be referred to, and c) order the collaboration

graphs from the most global to the most specific. By following these guidelines for the book trader system, the initial class cards shown in Figure 2.29 were revised as shown in Figure 2.32, each card describes a class.

Class: Book (concrete)
Superclasses: none
Subclasses: none
Hierarchy Graphs: none
Collaborations Graphs: Figure 2.31
Description: This class represents the books that are advertised in magazines and ordered by customer.
Contracts
3. Access book information
Know the book detail
sendBookDetail() returns book information
4. Update stock level
update(stock level) returns new value of stock level
Private Responsibilities
Record book information
Access book information

Class: Customer (Concrete)
Superclasses: none
Subclasses: none
Hierarchy Graphs: none
Collaborations Graphs: Figure 2.31
Description: This class represents the buyers who order the advertised books.
Contracts
2. Record customer information
Know the customer detail
recordCustomer(name, address) returns OK

Class: Advertisement (Concrete)
Superclasses: none
Subclasses: none
Hierarchy Graphs: none
Collaborations Graphs: Figure 2.31
Description: This class represents the advertisements that show the sale details of books.
Contracts
6. Calculate sum of response orders
sumOrder() returns number of orders
Private Responsibilities
Record advertisement information
uses Selling price (7)
Access advertisement information
use Order (5)

Class: Enquiry (Concrete)
Superclasses: none
Subclasses: none
Hierarchy Graphs: none
Collaborations Graphs: Figure 2.31
Description: This class represents the enquiries about the number of orders responding to an advertisement or about the progress of a specific order.
Private Responsibilities
Accept enquiry
Answer enquiry
uses Order (1)
uses Advertisement (6)

Class: Order (Concrete)
Superclasses: none
Subclasses: none
Hierarchy Graphs: none
Collaborations Graphs: Figure 2.31
Description: This class represents the orders of books that are provided by customers.
Contracts
1. Access order state
Know the order state
orderState() returns current state
5. Access order information
Know the order information
sendOrderDetail() returns order information
Private Responsibilities
Record order information
uses Customer (2)
uses Book (3)
Produce receipt
uses Book (4)
Update order state
Clear payment

Class: Selling price (Concrete)
Superclasses: none
Subclasses: none
Hierarchy Graphs: none
Collaborations Graphs: Figure 2.31
Description: This class represents the selling prices that are published for advertised books.
Contracts
7. Access the amount
amountPrice() returns current value of amount

Figure 2.32 Revised Class Cards for the Book Trader System

(3) Writing a Design Specification for Each Contract

A contract is recorded in a contract card. Seven contract cards that record the contracts defined in the book trader system are listed in Figure 2.33.

Contract 1:	Access order state
Server:	Order
Clients:	Enquiry
Description:	This contract allows clients to access the current state of an order

Contract 2:	Record customer information
Server:	Customer
Clients:	Order
Description:	This contract allows clients to record the details of a customer.

Contract 3:	Access book information
Server:	Book
Clients:	Order
Description:	This contract allows clients to access the details of a book.

Contract 4:	Update stock level
Server:	Book
Clients:	Order
Description:	This contract allows clients to update the number of the copies of a book.

Contract 5:	Access order information
Server:	Order
Clients:	Advertisement
Description:	This contract allows clients to access the details of an order.

Contract 6:	Calculate sum of response orders
Server:	Advertisement
Clients:	Enquiry
Description:	This contract allows clients to calculate the number of orders responding to an advertisement.

Contract 7:	Access the amount
Server:	Selling price
Clients:	Advertisement
Description:	This contrct allows clients to access the value of amount.

Figure 2.33 The Contract Cards for the Book Trader System

2.2.4.4 Products Generated

The products of the book trader system produced by the Wirfs-Brock method consist of (a) class cards, (b) a collaboration graph, and (c) contract cards.

2.3 Assumptions about analysis

Based on the distinction drawn between object-oriented analysis and design described in Section 1.1.2, these four methods can be viewed as analysis methods since the above study

shows that they all do analysis during system development. Even though the Booch method and the Wirfs-Brock method do not give a clear milestone between analysis and design, they still analyse a problem domain or an application scenario in order to produce an analysis model that is deliverable to design. By studying these methods, the assumptions of the methods about analysis can be explored. These assumptions show what a method is basically supposed to do for analysis, which is useful for us in establishing the required framework in the next chapter.

2.3.1 Assumptions in the OOA Method

The claims of the OOA method indicate that it is an analysis method that aims to build an object model of a system using the principles that are supposed to be significant to analysis. This object model consists of five layers each of which emphasises the concepts such as 'object' and 'structure' and includes the elements such as 'class-&-objects' and 'attribute'. A process of analysis, composed of five activities, is provided by this method in order to build an object model. The use of this method to analyse the book trader scenario shows that each activity includes several actions, such as 'where to look', during analysis. It also shows that the method gives guidelines (e.g., 'where and what to look for') to assist each action and the criteria (e.g., 'needed remembrance') to help in the determination of the significant elements in an object model.

This study finds that the inputs of analysis may be a description of a problem domain or may be a specific scenario, and the products of analysis include a five-layer object model and the specifications of objects and classes. It also reveals that the process of analysis is data-driven since the operations are primarily defined upon the attributes (i.e., data) in the class-&-objects, when we built the object model for the book trader system. The object model is represented by the OOA notation, object state diagrams and service charts.

2.3.2 Assumptions in the OMT Method

As shown in Section 2.2.2, the OMT method includes the analysis stage of system development. This method addresses five themes that it claims are significant and important in object orientation. The method builds three kinds of model during analysis: the object model, the dynamic model and the functional model. An iterative process is provided in order to construct these models.

The study of this method shows that the main input of analysis is the description of a problem, such as the book trader scenario, and the three models are built by analysing this input using this method. These three models emphasise the concepts such as 'object structure' and 'object behaviour', and they consist of a set of elements, e.g., 'association' and 'event'. In particular, an object model describes the structure of a system (such as Figure 2.23), a dynamic model specifies the behaviour of an object in the system (such as Figure 2.18); and a functional model represents the transitions of values from input to output in the system (such as Figure 2.20).

The study also shows that the process of analysis provided by this method consists of five steps such as 'constructing the object model'. Each step includes a sequence of substeps such as 'identifying objects and classes'. In this method, the data structure is emphasised more than the process structure since the method assumes that the former is more stable than the latter. An object model is thus the basic model and the other two models are built upon it. The study also reveals that the process of analysis in this method is data-driven since it specifies the objects, classes, their attributes and their structures in an object model first and the behaviour and functions of objects and classes are then specified in terms of the object model.

Additionally, a set of guidelines and criteria is used when using this method to carry out the steps and substeps in the process of building the three kinds of model. For example, one guideline proposes looking for the nouns and noun phrases from a given scenario, and the criteria such as 'the significant class should not be redundant in the object model' are used to choose the significant classes in the construction of an object model.

The products of analysis generated by the OMT method include three different models that are represented by the extended entity relationship diagrams and a data dictionary, statecharts and event flow diagrams, and data flow diagrams.

2.3.3 Assumptions in the Booch Method

Booch claims that his method supports the development of systems from analysis through design to implementation, and it emphasises the discovery of objects and classes from a problem domain and the invention of objects and classes on a computer domain [Booch91]. Four principles are used by this method, as addressed by most object-oriented programming languages. To specify objects and classes, the method aims to construct an object model by analysis based upon the four principles. A 'round-trip gestalt' process, i.e., analyse a little, design a little, is proposed for building this object model, and it includes four steps.

The Booch method requires the identification of objects and classes from a problem domain using any technique, which may be derived from some other analysis method. For example, we used the OOA method. Then it identifies the semantics of these objects and classes, such as their operations and relationships, from the problem domain in the 'round-trip' process. This shows that the Booch method should be considered as an analysis method since it is concerned with the analysis of a problem domain as part of system development.

The study of the Booch method shows that the process of analysis in this method emphasises the identification and specification of the operations for the objects and classes, since they have strong impact on the identification and specification of other elements such as the relationships between objects or classes in an object model. However, no detailed guidelines and criteria are given to assist the steps of this process, leading to difficulties in the selection of the significant elements in an object model using this method. An object model for this system is mainly represented by a class diagram, state transition diagrams and an object diagram.

2.3.4 Assumptions in the Wirfs-Brock Method

From the claims given in Section 2.2.4, the Wirfs-Brock method basically aims to design object-oriented systems. The study, however, shows that this method also does analysis in order to identify objects and classes and their responsibilities from a complete specification of system requirements. In fact its authors regard the real word as a system and so an application scenario could be used as a specification of system requirements in this method. Consequently, this method is also an analysis method. Five issues in particular are addressed by this method. Based upon these issues, an object-oriented system is modelled and specified. The object model of the system focuses strongly on the description of the responsibilities of classes and the collaborations between the classes or the subsystems in the system. A process of analysis is provided by this method to construct an object model for an application, involving two phases with five steps.

The study of this method in Section 2.2.4 shows that a specification of system requirements or an application scenario is the input of analysis. A model is produced by a process that consists of five steps such as 'finding classes'. Each step also includes a set of substeps such as 'looking for noun phrases'. Guidelines and criteria are given to assist these steps and substeps. For example, classes can be abstracted and selected from noun phrases by following criteria such as 'keep physical objects'.

This study shows that the process in this method is responsibility-driven, or process-driven, and the responsibilities of classes for a system have impact on identifying and defining other elements in the model, such as 'contract' and 'collaboration' between the classes. Attributes of the classes are not explicitly represented in the model of the system. The products of analysis are represented by a collaboration graph, class cards, and contract cards.

Chapter 3

The Framework

This chapter defines a *framework* for assessing object-oriented analysis methods and shows how such a framework may be used as a comprehensive basis for understanding object-oriented analysis methods. In this chapter, Section 3.1 discusses the essential features of object-oriented analysis methods, Section 3.2 defines the framework, Section 3.3 and 3.4 present more detail and examples of each component in the framework, and finally Section 3.5 describes the process of assessing an analysis method in terms of the framework defined.

3.1 Essential Features of Object-Oriented Methods

3.1.1 Basic Assumptions about Object-Oriented Analysis

Four representative analysis methods [Coad91a, Rumbaugh91, Booch91, Wirfs90] were applied and discussed in the previous chapter. That chapter discussed the assumptions made about object-oriented analysis by each method, as described in Section 2.3. The study also showed that some of the assumptions are common and basic to object-oriented analysis, being crucial to the success of such analysis. These basic assumptions include:

(1) the *inputs* required for analysis, such as a problem domain,

(2) the *models* aimed to be built during analysis, specifying what a system should do,

(3) the *notations* used for representing these models,

(4) the *process* of analysis provided for building the required models, and

(5) the *products* generated from analysis.

A study and examination of other analysis methods (e.g., Synthesis [Page89], OOA [Shlaer88 and 92], OOSE [Jocobson92], Ptech [Martin92a], OBA [Rubin92], OOSA [Embley92], BON [Nerson92], HOOD [Robinson93], MOOD [Capretz93], Fusion [Coleman94], and Syntropy [Cook94a]) reveals similar basic assumptions. For example, OOA [Shlaer88 and 92] can be said to have the following assumptions: *input* (i.e., a problem domain), *analysis models* (an information model, a state model, and a process model), *notations* (domain chart, information structure diagram, state transition diagram, and action data flow diagram), a *process of analysis* (step 1: develop the information model, step 2: develop the state model, step 3: develop the process model, etc.), and *products of analysis* (the three above models, supporting tables, descriptions, and lists).

These basic assumptions that can be said to exist in analysis methods represent the essential features of the methods, according to our research, and the particular nature of these features determine the similarity and/or difference between methods. Analysing and assessing these features of an analysis method would be a useful way of acquiring an understanding of the nature of a method.

3.1.2 Essential Features of Analysis Methods

Except for the assumptions shown above, we also found that the models produced by the four methods were different even though they described the same problem scenario used in the study. A further examination of the methods shows that the object concepts — such as 'object' and 'inheritance relationship' and 'object behaviour'—emphasised by the methods have strong impact on the configuration of the models. That is, different concepts, or even the concepts that look similar to one another, may make the content of the models different. For instance, the concept 'message' is emphasised in the OOA method and so its model includes an element 'message connection'; while the concept 'client-server' is emphasised in the Wirfs-Brock method and thus 'client-server contract' is provided as an element in the model. More examples of such impact can be found in the applications shown in the previous chapter. On the other hand, the same concept may be

implemented in different ways by different analysis methods. For example, the concepts 'object' and 'class' are implemented by the elements 'class-&-object', 'attributes' and 'service' in the OOA method, and by the elements 'object', 'class', 'object state' and 'operation' in the object model in the Booch method. According to the definitions of the elements in both methods, they have different meanings and play different roles in analysis [Liang94].

In addition, further examination shows also that the principles, such as 'abstraction' and 'encapsulation', used by an analysis method often make the method distinctive and different. Distinguishing the various meanings of such principles will be helpful in explaining why some analysis methods focus on similar or different concepts within analysis. For example, the OOA method claims that the principle 'abstraction' particularly refers to 'data abstraction' and therefore the concept 'object' is a notion of something in a problem domain that is specified as an encapsulation of attribute values and their exclusive services acting on the attributes. In contrast, the Wirfs-Brock method regards this principle as 'one of process abstraction' so that the concept 'object' is a notion of a conceptual entity in the real world that is an encapsulation of functions and data which are maintained by the functions. Different interpretations of the principle 'abstraction' in these two methods make the meaning of the concept 'object' different between the two methods. These principles and concepts are the basis for object modelling and are fundamental to analysis. As well as these underlying principles and concepts, an essential feature of an analysis method is the general approach taken during analysis to the construction of object models. We term this general approach the 'tactic of analysis'. For example, the tactic of analysis in the OOA method is 'data-driven', whilst the tactic of analysis in the Wirfs-Brock method is 'responsibility-driven'.

To summarise, the extra features below are also essential in an analysis method and will be important and useful in the understanding of a method.

(6) the *fundamental concepts* underlying the various models, such as object;

(7) the *fundamental principles* informing these concepts, such as encapsulation; and

(8) the *tactic of analysis*, such as data-driven.

3.2 Definition of the Framework

This section defines a framework, based on the essential features identified above, for assessing individual analysis methods. Such a framework will form a comprehensive basis for understanding and comprehending individual analysis methods.

3.2.1 'What' and 'How' Aspects of Analysis Methods

Booch [Booch 91] emphasises that object-oriented analysis includes the process of identifying and modelling the essential objects and classes and their logical relationships and interactions. Furthermore, he states that, no matter which analysis method is employed, the important factor is that the products of analysis provide us with a complete enough model of the problem from which we may begin to design a solution. This distinction between 'model' and 'process of analysis' is important since generally the model represents *what* the method aims to do and the process shows *how* the method can reach its aim. Accordingly, all the essential features of an analysis method are catalogued into two basic aspects as follows:

(1) the 'what' aspect, i.e., what *the method intends to do in analysis,* and

(2) 'how' aspect, namely, how *the method does what it intends to do in analysis.*

In general, the essential features *model* and *fundamental principle* and *fundamental concept* are included under the 'what' aspect of the method, as a *model* is concerned with what analysis specifies as being required and *fundamental principle* and *fundamental concept* are a basis of *model.* The essential features *notation, tactic of analysis, input, process,* and *product* of analysis are classified into the 'how' aspect of the method according to their major roles in analysis. The framework is defined below by focusing on these two aspects of analysis methods.

3.2.2 The Definition of the Framework

A *framework* for assessing object-oriented analysis methods is defined as follows:

Framework

The framework for assessing an analysis method consists of two parts that correspond to the two aspects of the method:

(1) the *principle part,* which focuses on the 'what' aspect of an analysis method. This part includes the following components: 'fundamental principle', 'fundamental concept', and 'model', including type and element. These components are used to assess the essential features in the 'what' aspect of the method.

(2) the *practice part,* that emphasises the 'how' aspect of an analysis method. This part includes the following components: 'notation', 'tactic of analysis', 'input of analysis', 'process of analysis', with steps, guidelines and criteria, and 'product of analysis'. These components are used to assess the essential features in the 'how' aspect of the method.

This framework can be represented graphically in a summary fashion by a hierarchical table (see Table 3.1). In the table, each column at the first (highest) header level contains each part of the framework (principle or practice), each column at the second header level has one component in a part, and each column at the third level includes a subset of a component, where appropriate. In terms of the framework, the features identified by assessing an analysis method will be represented in cell rows under the columns of Table 3.1 (for example, see Table 3.2 (a) and (b)). However, it must be emphasised that this table is just a convenient *graphical representation* of the framework. The framework itself is a detailed approach including the interpretations of its parts and components and a process of using it to assess analysis methods, as is now described.

Method / Stage	Principle Part				Practice Part						
	Fundamental Principle	Fundamental Concept	Model		Notation	Tactic of Analysis	Input	Process of Analysis			Product
			Type	Element				Step or Activity	Substep or Action	Guideline & Criterion	
Analysis											

Table 3.1 A Framework for Assessing Object-Oriented Analysis Methods

Practice Part / Stage	Notation	Tactic of Analysis	Input	Process of Analysis			Product
				Step or Activity	Substep or Action	Guideline & Criterion	
Analysis	•Diagram —Object Diagram —State Transition Diagram —Data Flow Diagram •Text —Specification language •Table or card	Analyse a problem domain by Data-driven	• problem statements	•Build the object model •Build the behavioural model •Build the functional model	in each step (or activity) •Identify... •Find... •Define... •Construct...	•Find nouns from requirement statements •Attribute is not an object	•Object Model •Behavioural Model •Functional Model • Class Card

Table 3.2(a) Examples of the Practice Part

Principle Part / Stage	Fundamental Principle	Fundamental Concept	Model	
			Type	Element
Analysis	• Abstraction • Encapsulation	• Object • Relationship	• Object Model	• Object • Class • Attribute • Association
			• Behavioural Model	• State • Event • Action
			• Functional Model	• Process • Data flow • Data store

Table 3.2(b) Examples of the Principle Part

3.3 The Principle Part

3.3.1 Fundamental Principle

The word 'principle' is defined as follows in the dictionary [Oxford88]:

> **Principle**: 1) basic truth or general law of cause and effect; 2) guiding rule for behaviour; 3) general law shown or used in the working of a machine.

The fundamental principles of an analysis method are the basic 'laws' of analysis for the method and underpin the concepts and the construction of models within a method. When assessing an analysis method, we need to understand the content and meaning of these principles in order to understand why the principles are emphasised and how they may effect other features. However, it should be noted that the fundamental principles may not be explicit in a method (for example, this is the case with [Shlaer88 and 92]) and they may need to be abstracted from other features, such fundamental concepts. The following are examples of part of the fundamental principles used in the OOA method [Coad91a] and the OMT method [Rumbaugh91]:

- *Abstraction*

  Abstraction is a principle that can help to choose certain things over others [Coad91a]. Data abstraction is in particular used in this method.

- *Encapsulation*

  Encapsulation (also known as information hiding) is a principle that separates the external aspects of an object from the internal ones, such that only the external aspects can be seen and be accessed by other objects [Rumbaugh91].

• *Communication with messages*

This is a notable principle for interactions between objects.

With traditional methods, the principle 'abstraction' is classified into three categories, according to Olle's work [Olle88], corresponding to the three common aspects of a system: (a) *data abstraction* (that is, a data oriented perspective) which enables methods to emphasise and specify the data structure of a system; (b) *process abstraction* (that is, a process oriented perspective) by which methods can focus on and specify the supposed functions of a system without temporal concerns; and (c) *behaviour abstraction* which enforces methods to focus on and specify the time dependent (or temporal) processes of a system.

With object-oriented analysis methods, our research has shown that the principle 'abstraction' can also be used to focus on one or more aspects (i.e., data, process, and/or behaviour) of a system by specifying the static structure, operations and collaboration, and dynamic behaviour of objects within a system.[1] The principle 'abstraction' as used in object-oriented methods, therefore, is classified as follows:

(a) *abstraction with data,* by which an analysis method focuses on and specifies the static structure of objects within a system. Such abstraction is concerned with the attributes of objects and the relationships between objects.

(b) *abstraction with process,* by which an analysis method emphasises object behaviour within a system without temporal concerns. Such abstraction is concerned with the operations of objects and the functional collaboration between objects.

(c) *abstraction with behaviour,* by which an analysis method focuses on and specifies object behaviour within a system with temporal concerns. Such abstraction is concerned with the sequence of states and operations of objects over time.

[1] More detail of the discussion on this issue was shown and discussed in our two published papers [Liang93 and 94], for instance, the comparison of OOA methods and traditional analysis methods such as SSADM [Ashworth90] and the impact of these three perspectives on object modelling.

According to this classification, for example, the abstraction in the OOA method can be regarded as abstraction with data and behaviour, while the abstraction in the OMT method can be regarded as abstraction with data, process and behaviour (see Chapter 5). Other published work also shows how different types of abstraction may be used to emphasise different perspectives within a system using object-oriented methods. For example, Fowler [Fowler91], compares object-oriented methods by examining the abstraction in terms of a data view, a behavioural view and an architectural view (i.e., process view) of a system in object modelling.

3.3.2 Fundamental Concept

The word 'concept' is defined generally by the dictionary [Oxford88] as follows:

> **Concept**: idea underlying a class of things; general notion.

The fundamental concepts are the basic ideas behind the business of analysis in a method, and are underpinned by the fundamental principles used in a method. A fundamental concept may be explicitly declared or be implicit within a method (usually within the models built). A typical example of a fundamental concept is 'object':

- *Object*
  - — An object is an abstraction of something in a problem domain, reflecting the capabilities of a system to keep information about it, interact with it, or both [Coad91a].
  - — An object combines both data structure and behaviour in a single entity. It is a concept, abstraction, or thing with crisp boundaries and meanings for the problem at hand [Rumbaugh91].
  - — Object is the concept that means an entity that encapsulates both functions and data and that has a public interface and a private representation in order to make the internal details invisible to other entities [Wirfs90].

Other examples are as follows:

- *Class*
  - — Class is the concept that means a generic specification for a set of similar objects which share the same behaviour [Wirfs90].
  - — A class often is a description of one or more objects with similar properties [Coad91a].
- *Object behaviour*
  - — Object behaviour is a concept that means that an object has its own behaviour [Rumbaugh91].
  - — Object behaviour is the concept that expresses the interactions between objects, i.e., how an object acts and reacts in a system [Booch91].
- *Object structure*

  Object structure is a notion to illustrate how different objects collaborate with each other in terms of the outside views of the objects [Booch91].
- *Aggregation*

  An aggregation is a union of several objects [Jacobson92].
- *Grouping*

  Grouping is a notion of collecting the objects which are tightly related to one another within a system [Coad91a].
- *Partitioning*

  Partitioning is a notion of decomposing a large system into smaller parts [Rumbaugh91].

3.3.3 Model

In order to describe what a system needs to do to meet the requirements of an application scenario or a problem domain, a method must build (at least) an object model. In order to understand the detail of the individual models built by various methods, we need examine the various types of model employed and their constituent elements.

3.3.3.1 Type of Model

An analysis method may or may not aim to build more than one model in analysis. For example, as shown in Chapter 2, the OMT method [Rumbaugh91] builds three models of a system during analysis to correspond to each of three aspects of the system: the object model, the dynamic model and the functional model. In contrast, the OOA method [Coad91a] only builds one object model for a system, combining data and behaviour aspects into the one model. It is thus important to clarify the type of model in a method in order to understand its role and significance in the method. For example, the three types of model that are defined by the OMT method are:

(1) The Object Model

An object model describes the data aspect of a system: objects, classes, their attributes, their relationships, and so on.

(2) The Dynamic Model

A dynamic model represents the behaviour aspect of a system: states of objects, events, actions over time, interactions between objects, and so on.

(3) The Functional Model

A functional model describes the process aspect of a problem: inputs, outputs, processes, data flows, and so on.

3.3.3.2 Elements in a Model

A model consists of a set of elements each of which represents a part of a system. Different types of model normally include different kinds of elements, and more importantly, different methods may give different definitions of an element (for example, 'object'). So, in a method such as the OMT method, which emphasises the data aspect of a system, the element 'object' is typically defined with the attributes and the operations that act on the attributes; whereas a method such as the Wirfs-Brock' method, which

emphasises the process aspect of a system, this 'object' element is typically defined with the operations that imply the data accessed by those operations. A model therefore has to be understood and assessed by analysing the individual elements within it.

The following are examples of elements that come from the models in the OMT method [Rumbaugh91]:

(1) Elements in Object Models

The object model includes elements such as objects, classes, attributes and links between objects as follows.

a) *Object*

An object (or object instance) has attributes, states, and operations. All objects are distinguishable by their identities.

b) Class

A class is a group of objects which have common attributes and operations. A class represents the common definition of the objects. In addition, a class that has no object is called an 'abstract' class.

c) Attribute

An attribute is a data value, not an object, held by the objects in a class.

d) Link

A link is a physical or conceptual relationship between objects.

(2) Elements in Behaviour Models

The behaviour model includes elements such as events, states and activities as follows:

a) *Event*

An event is something that happens at a point in time.

b) Activity

An activity is associated with a state and it is an operation that takes time to complete.

c) State

An object state is an abstraction of the attribute values and links held by an object.

(3) Elements in Process Models

The process model includes elements such as 'process' and 'data flow' as follows:

a) *Process*

A process is an operation which transforms data values.

b) Data Flow

A data flow connects the output of an actor or process to the input of another actor or process.

3.4 The Practice Part

In the practice part of the framework, five components—notation, tactic of analysis, input of analysis, process of analysis, and product of analysis—are included, as shown in Table 3.1.

3.4.1 Notation

The models in the principle part are represented by notations in the practice part. A method may create its own notation or use some existing notation for this purpose. A notation may be graphical or textual or a mixture of them. For example, extended entity-relationship diagrams are used to represent the object model in the OMT method [Rumbaugh91] and object state diagrams are used to express the states of objects in the object model in the OOA method [Coad91a].

In an analysis method, it is the model rather than the notation that reflects the fundamental concepts that are addressed by a method. This means the same model could be represented by different notations. However, because a model is often introduced together with the notation in some methods, the notation may be perceived to be equal to the model. This may lead to a misunderstanding of the roles of both model and notation in a method, causing people who do not like the notation to reject the use of the model even if they may think that the fundamental concepts in the method are highly desirable for object-oriented analysis. The framework therefore distinguishes between 'model' and

'notation' in order to avoid confusion in the understanding of a method. The examples of the notations used by analysis methods are shown in Table 3.3, by referring to the models which they may represent.

Models / Notations	Object Model	Behaviour Model	Process Model
Diagrams	Extended Entity-Relationship Diagram Object Diagram Class Diagram	State Transition Diagram Event Trace Diagram	Data Flow Diagram
Texts	Data Dictionary Class Card Specification Template		Process Description Language

Table 3.3 Examples of Notations

3.4.2 Tactic of Analysis

A tactic of analysis in a method is the general approach to the process of analysis and it provides the basis for determining the process of analysis and, in particular, the sequence of the steps and substeps. For example, two differing tactics of analysis are illustrated by the OOA method and the Wirfs-Brock method, as follows:

(1) Data-Driven

The OOA method uses a 'data-driven' tactic of analysis, as claimed by its authors. With this tactic, the attributes in objects and classes are identified prior to the operations performed on these attributes.

(2) Responsibility-Driven

The Wirfs-Brock method uses a responsibility-driven tactic of analysis. Attributes are not specified explicitly in the model of the system in the process of analysis.

3.4.3 Input of Analysis

The input of analysis required by an analysis method could be any form of system requirements. For example, the OOA method [Coad91a] allows an incomplete input (i.e., incomplete system requirements) at the beginning of analysis and then a complete one will be obtained gradually by interviewing users during analysis. In contrast, the Wirfs-Brock method requires a complete input (i.e., complete system requirements) before starting analysis.

3.4.4 Process of Analysis

The process of analysis in a method commonly consists of a set of steps/substeps or activities/actions, containing guidelines and criteria for analysis. In order to understand the process of analysis, the steps (or activities) and substeps (or actions) as well as the guidelines and criteria should be analysed and assessed.

3.4.4.1 Step or Activity

A step or an activity usually covers an independent part of analysis in the process of analysis. For example, the process of analysis in the OMT method [Rumbaugh91] consists of three steps as follows:

Step 1. constructing the object model
Step 2. constructing the dynamic model
Step 3. constructing the functional model.

3.4.4.2 Substep or Action

A step or an activity may be partitioned into several substeps or actions each of which defines part of a model. Furthermore, if a substep or an action is still too complex to carry out, a further partition of this substep or action is done until the final substep or action is simple enough to carry out. The step '*constructing an object model*' above, for example, is partitioned into the substeps in the OMT method as follows:

Substep 1. identify objects and classes
Substep 2. prepare a data dictionary

Substep 3. identify associations
Substep 4. identify attributes
Substep 5. identify inheritance, etc.

The correspondence between this step and its substeps is shown in a hierarchical form in Table 3.4.

Step	Substep
Step 1. Constructing the object model	Substep 1. Identifying objects and classes
	Substep 2. Prepare a data dictionary
	Substep 3. Identify associations
	Substep 4. Identify attributes
	Substep 5. Identify inheritance
	etc.

Table 3.4 Example of a Step and Its Substeps

3.4.4.3 Guidelines and Criteria

Guidelines enable the analyst to carry out a step/substep or an activity/action with confidence. In the OMT method, for example, the objects and classes are identified using the guideline of abstracting the nouns in the problem statements. Criteria are the rules or heuristics provided by an analysis method for selecting and specifying the correct elements in the models of a system. For instance, the OOA method provides as the criteria for selecting the significant class-&-objects in the activity 'finding class-&-objects': the significant class-&-objects should be a) needed remembrance (the authors' term) b) needed behaviour, c) multiple attributes, d) more than one object in a class, e) always applicable attributes, f) always-applicable services, g) domain based requirements, and h) not merely derived results. Generally, guidelines show the right way to do analysis and criteria define the right things for a system.

3.4.5 Product of Analysis

The product of analysis is the deliverable from analysis to design in an analysis method. The previous chapter showed that the products of analysis often include the models built

through analysis and the specification documents of individual objects and classes. For example, the following products of analysis are produced by the process of analysis in the OMT method:

- *an object model* — object diagrams
- *a dynamic model* — state diagrams
- *a functional model* — data flow diagrams.

The previous chapter showed that different analysis methods may produce different products from analysis, in particular, when they define different models, or use different notations, or provide different guidelines and criteria for analysis.

3.5 The Process of Using the Framework to Assess an Object-Oriented Analysis Method

The assessment of an analysis method using the framework needs to focus not only on the clarification of the content of the two aspects (the principle part and the practice part) but also on the co-operation between the two aspects, in order to gain a comprehensive and deeper understanding of the nature of the method and to interpret the features in the two aspects accurately. The process of using the above framework in the assessment of an OOA method is shown in Figure 3.1.

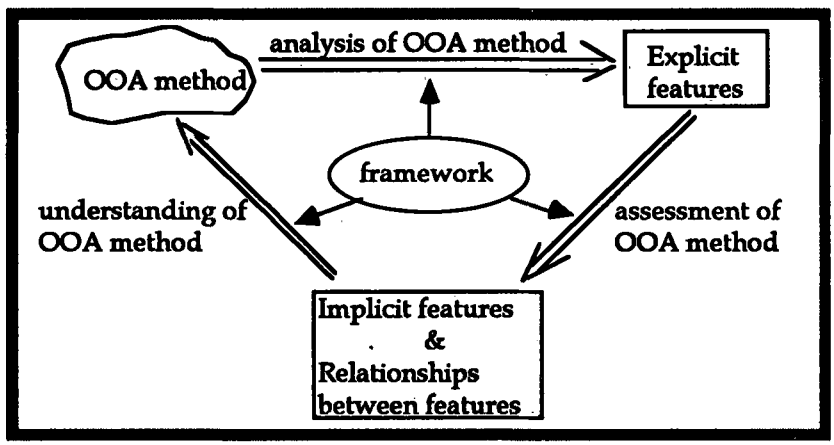

Figure 3.1 Process of Using the Framework to Assess an Analysis Method

There are two stages in the process: (1) analyse the method by identifying the explicit features of the method, and (2) assess the method through clarifying the content of the features identified and considering the relationships between them. The implicit features in the method may be identified using the relationships considered at the second stage. Each stage consists of a collection of activities (with actions) and includes criteria for analysis and assessment, represented as a series of questions. The features of the method and their relationships are assessed against the criteria offered and the results are recorded in Table 3.1. Table 3.5 shows the detail of the process in a hierarchical tabular form.

3.5.1 Stage 1: Analyse the Method

At this stage, a method is analysed by focusing on its 'what' and 'how' aspects in terms of the framework, and the essential features that reflect the two aspects of the method are identified, corresponding to the components in the principle and practice parts of the framework and are then recorded in the Table 3.1. Our experience shows that some features (e.g., 'model', 'notation' and 'process of analysis') of an analysis method are often explicit while others (for instance, 'fundamental principle' and 'fundamental concept' and 'tactic of analysis') may be implied in the text of the method. Those essential features which are explicit in the method are identified directly from the text of the method. However, the implicit features may not be recognised until assessing the features identified and their dependencies and relationships at the next stage. The activities included in this stage are as follows.

3.5.1.1 Activity 1.1: Analyse the 'What' Aspect of the Method

The first activity at this stage is to analyse the 'what' aspect of the method, i.e., what the method intends to do for object-oriented analysis, in terms of the principle part of the framework in sections 3.2 and 3.3. Three actions are used to identify the three essential features—'fundamental principle', 'fundamental concept', and 'model'— of the method, corresponding to the three components in the principle part of the framework.

Stage	Activity	Action	Criterion
Stage 1: Analyse the method	Activity 1.1: Analyse 'what' aspect	Action 1.1.1: Identify fundamental principles	(a), (b), (c), (d)
		Action 1.1.2: List fundamental concepts	(a), (b), (c)
		Action 1.1.3: Analyse models	(a), (b), (c), (d), (e), (f)
	Activity 1.2: Analyse 'how' aspect	Action 1.2.1: Illustrate the notation	(a), (b), (c), (d)
		Action 1.2.2: Identify the tactic of analysis	(a), (b)
		Action 1.2.3: List the input of analysis	(a), (b), (c)
		Action 1.2.4: Describe the process of analysis	(a), (b)
		— identify steps/substeps or activities/actions	(a), (b), (c)
		— collect guidelines and criteria	(a), (b)
		Action 1.2.5: Identify the product of analysis	(a), (b), (c), (d)
Stage 2: Assess the method	Activity 2.1: Assess 'what' aspect	Action 2.1.1: Assess the relationship between the features in the 'what' aspect	
		— Assess the relationship between 'fundamental concept' and 'model'	(a), (b)
		— Assess the relationship between 'principle' and 'concept'	(a)
		— Classify the principle 'abstraction'	(a), (b), (c)
		— Check the 'pyramid' construction of the features described in the table	(a), (b), (c)
		Action 2.1.2: Assess the content of the 'what' aspect	(a), (b)
	Activity 2.2: Assess 'how' aspect	Action 2.2.1: Assess the relationship between the features in the 'how' aspect	
		— Assess relationship between 'notation' and 'process of analysis'	(a), (b), (c)
		— Assess relationship between 'input' and 'process of analysis'	(a), (b), (c)
		— Assess relationship between 'tactic' and 'process of analysis'	(a), (b), (c)
		— Assess relationship between 'product' and 'process of analysis'	(a), (b), (c)
		Action 2.2.2: Assess the content of the 'how' aspect	(a), (b)
	Activity 2.3: Assess relationship between two aspects	Action 2.3.1:Assess relationship between 'model' and 'notation'	(a), (b), (c), (d)
		Action 2.3.2:Assess relationship between 'model' and 'tactic of analysis'	(a), (b), (c)
		Action 2.3.3:Assess relationship between 'model' and 'process of analysis'	
		— assess the relationship between 'model' and 'step/substep'	(a), (b), (c), (d), (e)
		— assess the relationship between 'model' and 'guideline and criteria'	(a), (b), (c), (d)
		Action 2.3.4:Assess relationship between 'model' and 'product'	(a), (b), (c), (d)

Table 3.5 The Process and Criteria for Assessing an Object-Oriented Method

Action 1.1.1: Identify fundamental principles

Based on the account of the fundamental principles in section 3.3.1, the following criteria may be used to identify the fundamental principles used in an analysis method:

(a) Are the principles usually employed by most analysis methods also used in this method? And how are they defined in the method?

This criterion provides a way of detecting the principles used in the method, based upon experience and knowledge of the typical principles which are often used in many analysis methods. Our experience and knowledge shows that the terms, such as 'abstraction', 'encapsulation', 'information hiding', 'inheritance', and 'scale', appearing in a description of an analysis method, often represent the potentially fundamental principles of the method.

(b) Is there any other law which is particularly regarded as fundamental to object-oriented analysis in the method? And what does it mean in the method?

This criterion focuses on finding the principles which may be used only in this method. These principles may be claimed explicitly and defined in the text of a description of the method. For example, 'communication with messages' is claimed as a principles of analysis in the OOA method [Coad91a] (See section 4.1.1).

(c) Why does the method regarded these principles, rather than others which you may know from another analysis method, as fundamental to object-oriented analysis?

The author of an analysis method often explains the reasons why it is necessary and important for the method to use these principles. This may be helpful to consider the significance of the principles used in the method and to understand the meaning of the principles accurately.

(d) For each principle identified, is it fundamental and necessary to the success of object-oriented analysis per se*?*

Our experience shows that an analysis method may use object-oriented analysis principles and object-oriented design principles without distinguishing between the two. Disentangling the analysis principles from the design principles then becomes important and useful. Of course, this may be particularly the case where the method also supports object-oriented design. For example, 'polymorphism' is used as a fundamental principle in the Wirfs-Brock method in order to support the design of a generic structure in a system. This principle, therefore, is not an analysis principle according to the criteria here. It is ignored in the assessment of this method (see section 4.4.1).

The principles identified in this action and their definition or descriptions should be recorded in the first column in the principle part of Table 3.1.

Action 1.1.2: Identify fundamental concepts

Fundamental concepts are discussed in section 3.3.2. In this action the fundamental concepts defined in the text of a description of a method are identified according to the following criteria:

(a) Which of the concepts commonly defined in object-oriented methods are also used in this method?

Through the study of a large number of current object-oriented analysis methods, it is found that some concepts such as 'object', 'class', 'object structure', 'aggregation', 'inheritance relationship', 'association', 'object behaviour (or object life cycle)', 'message', and 'partition/grouping' appear in the methods quite frequently. This criterion enables to consider whether or not the method assessed also refers to or defines similar concepts.

(b) Is there any specific concept which is also regarded as fundamental to object-oriented analysis in the method and, if so, what is its definition?

In a similar fashion as a particular fundamental principle, the author of an object-oriented analysis method may define specific concepts which he or she considers fundamental to analysis. For example, the responsibility of an object is regarded as a fundamental concept in analysis in the Wirfs-Brock method, the term 'object responsibility' is found from, and defined in, the text describing this method and it represents the above concept (See section 4.4.1). The terms which are particularly defined and explained (in texts describing the method) as important concepts should be considered in this action, since they may be the concepts regarded as fundamental to analysis by the method.

(c) Why does the method regard these concepts as fundamental to object-oriented analysis?

This criterion is helpful to understand the content of these concepts and how they are utilised by the method.

All concepts identified in this action and their definitions should be recorded in the second column of the principle part in Table 3.1.

Action 1.1.3: Analyse models

The role of models in an analysis method has been discussed and elaborated in section 3.3.3. This action is to identify and analyse the models which will be built by the method. The following criteria may be used:

(a) How many different types of model are constructed in analysis by the method?

(b) Which aspect of a system (i.e., data, process, or behaviour) is emphasised and specified by each model upon objects?

(c) What is the relationship between identified models and how do they refer to one another, if there is more than one model in the method?

(d) What elements are included in each model to specify what a system needs to do without implementation detail? And which part of the system does each element represent?

(e) How is each element defined by the method and how is it connected with other elements in the same model?

(f) Is there any constraint or condition on the use of each element in each model?

The first three criteria emphasise the recognition of multiple models built by analysis and, in particular, which of these models is given any primacy by the method, as well as their roles and references to one another in the specification of a system. The last three criteria focus on the identification of the elements in each model and clarify their contexts, roles and constraints in the model.

In addition, the (linguistic) terms used to refer to elements should be examined carefully, as a term that refers to an element in several models (in different methods) sometimes may represent a dilemma in a method due to the lack of standards in object-oriented methods. This may lead to an incorrect interpretation and misunderstanding of

the elements in object-oriented methods; that is, elements with the same meaning may be Refer to by different terms or elements with different meanings may be Refer to by the same term in different methods. For instance, the element 'object' is represented by the same term *object* in both the OMT method [Rumbaugh91] and the Booch method [Booch91], but they actually have different declarations and meanings (see Chapter 2), although this element with different meanings supports and implements the same concept 'object' in these two methods. This example shows that it is important to assess the relationships between the fundamental concepts and the elements in models so that the nature of an analysis method can be recognised (see action 2.1.1). The definition of each term should be listed in a glossary as a reference for the method.

3.5.1.2 Activity 1.2: Analyse the 'How' Aspect of the Method

The second activity of the first stage focuses on the analysis of the 'how' aspect of the method (i.e., how the method carries out analysis) in terms of the practice part of the framework, discussed in sections 3.2 and 3.4. This activity includes five actions in which five essential features ('notation', 'tactic of analysis', 'input of analysis', 'process', and 'product of analysis') of the method are identified, corresponding to the components in the practice part of the framework.

Action 1.2.1: Illustrate the notation

Section 3.4.1 analyses the role of notation in an analysis method. The notation is normally presented and described in detail in the text describing the method. In other words, the notation can often be directly obtained by reading through a description of the method, as shown in Chapter 4. The following criteria may be useful for this purpose:

(a) What symbol is used for representing each element in each above model?

(b) Is there any constraint or condition on the use of a symbol?

(c) How many sorts of (textual or graphical) notation are used in the presentation of models in the method and what is the speciality of each sort of notation?

(d) why does the method choose to use such notation?

The various kinds of notation identified are recorded in the first column of the practice part in Table 3.1. Separate diagrams and description may be drawn and written as the documentation of the symbols, using all sorts of notation.

Action 1.2.2: Identify the tactic of analysis

The meaning of the tactic of analysis is explained in section 3.4.2. The tactic of analysis used in an analysis method is identified in this action by using the criteria as follows:

(a) Does the author of the method claim, or state something about, the tactic of analysis (e.g., data-driven or process-driven)?

(b) What does the tactic shown actually mean in the method?

If no claim about the tactic of analysis is directly found from the text describing the method, the identification of such a tactic could be delayed until the action 2.3.2 in section 3.5.2.3, where the tactic can be explicated using the relationship between models and the tactic of analysis. The tactic of analysis identified is documented in the second column of the practice part in Table 3.1.

Action 1.2.3: List the input of analysis

The textual description of an analysis method often shows the information about the input needed for analysis, as stated in section 3.4.3. To recognise the input of analysis in the method, the following criteria may be used:

(a) what kind of information (e.g., a problem situation or a problem domain) does the method regard as the input of analysis?

(b) where does the required information come from (for instance, from problem statements or from dialogue with users)?

(c) Must the input information be complete before starting analysis or, alternatively, is it allowed to be incomplete at the beginning of analysis?

These criteria enforce a consideration of the content of the input of analysis and the sources of the input. They also show the way a method collects the input information. The input information found is listed in the third column of the practice part in Table 3.1.

Action 1.2.4: Describe the process of analysis

The process of analysis is often demonstrated in the description of an analysis method by various application examples. Section 3.4.4 states that such a process usually consists of steps/substeps or activities/actions, as found in the study of Chapter 2. The process of analysis in a method may be identified according to the following criteria:

(a) How does the method do analysis in practice?

(b) Does the method provide a specific process for carrying out analysis?

The process identified should be further analysed in this action, in order to find the detail of the steps/substeps (or activities/actions) contained and guidelines and criteria involved in the process.

— *Identify steps/substeps or activities/actions in the process*

The criteria used here may be as follows:

(a) How many pieces of work needed to be completed by analysis are covered by the process in the method?

(b) Is the process decomposed into individual steps or activities such that each of them covers a piece of the work?

(c) Does the method further partition a step or an activity into substeps or actions so that each of them only covers a piece of the part of the work?

The steps/substeps or activities/actions identified are recorded in the column 'step or activity' in Table 3.1.

— *Collect guidelines and criteria provided in a step/substep or an activity/action*

To identify guidelines and criteria involved in the process, the following criteria may be used:

(a) Does the method give the guidelines that show the way of carrying out each step/substep or activity/action? And how they are described?

(b) Does the method offer criteria for choosing the right information for object modelling in each step/substep or activity/action? And what are their definitions?

The guidelines and criteria and their details identified are listed in the column 'guideline & criteria' in Table 3.1.

Action 1.2.5: Identify the product of analysis

Section 3.4.5 states that the product of analysis (i.e., the outcome of analysis) is the deliverable from analysis to design in system development. The following are the criteria of identifying such product in an object-oriented analysis method:

(a) Does the method indicate the product of analysis?

(b) Which result of analysis could be the deliverable to design according to the method?

(c) What does the deliverable consist of?

(d) Is there any constraint or condition on the product of analysis?

The product and detail identified are documented in the last column of the practice part in Table 3.1.

3.5.2 Stage 2: Assess the Method

Based upon the framework, the explicit features of an analysis method are identified in terms of the criteria provided in the previous stage and the details of them are recorded in Table 3.1. At this stage, the method is assessed by examining the content of its essential features and assessing the relationships between the features. The assessment of the relationships between the features, in addition, may explore the implicit features of the method, because of the dependency of one feature on another, as stated below.

3.5.2.1 Activity 2.1: Assess the 'What' Aspect of the Method

By analysing a method in terms of the framework, three kinds of essential features (i.e., 'fundamental principle', 'fundamental concept' and 'model') that reflect the 'what' aspect of the method are identified in the previous stage. However, the content of these features needs to be further clarified by examining their meanings and their relationships in order to avoid misunderstanding the method. The study of analysis methods shows that there exist dependencies between these three features. For example, the definition of an element such as 'class-&-objects' (see [Coad91a], for example) in a model is normally defined according to the definition of a fundamental concept such as 'object' and 'class' in the method, and this fundamental concept also refers to a fundamental principle such as 'abstraction' and 'encapsulation' in the method. This means that a chain of dependency exists between the features in the 'what' aspect of a method: 'model' depends on 'fundamental concept' and 'fundamental concept' depends on 'fundamental principle'. Such a chain can be represented by a 'pyramid' construction shown in Figure 3.2.

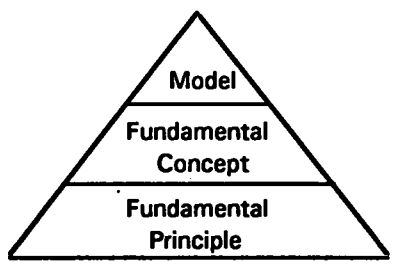

Figure 3.2 The 'Pyramid' Construction of Three Kinds of Essential Feature

In this activity the dependency between the features in the 'what' aspect of the method is considered and examined in order to understand the meanings of the features correctly.

Action 2.1.1: Assess the relationship between the features in the 'what' aspect

The relationship between the features recorded in the principle part of Table 3.1 is determined and assessed according to the 'pyramid' construction. The implicit features in the 'what' aspect of a method are also identified and described using the relationship identified.

— Assess the relationship between 'fundamental concept' and 'model'

The assessment of such relationships is also important to the reuse of the objects defined in different analysis methods. For instance, the study in Chapter 2 showed that the object *Book* was identified and defined by all four methods used since the methods all emphasise the fundamental concept 'object', but the elements 'Book' in their models had different declarations and meanings. When considering to reuse the object *Book* defined in the OMT model, we might discover that the object *Book* could not be directly reused by the Wirfs-Brock model without change because of its different definitions in the two models. Namely, these two methods use different elements in their models to represent and implement the similar fundamental concept 'object'. Whether or not an object could be reused in a model (in a different analysis method from its origin) depends on the nature of both the fundamental concepts and the elements in the models built by the two methods. The following criteria may be used for such assessment:

(a) For each element in the 'model' column, which concept in the 'fundamental concept' column does it represent and depend on?

According to the 'pyramid' construction, each element in a model may support and implement one or more fundamental concepts in the method. Usually the element is defined according to both the role which it plays in the model and the fundamental concepts which it supports and implements. For example, to support and implement the fundamental concept 'object' (meaning here the things important in a problem domain), the element 'object' is defined as 'an encapsulation of attribute values and their exclusive services' in the OOA model [Coad91a], or as 'an object has state, behaviour and identity' in the object model of the Booch method [Booch91].

(b) For an element which seems not to depend on any explicit concept in Table 3.1, does there exist an implicit concept which is supported and implemented by this element?

This criterion is very useful to identify the implicit concepts which are actually used but not declared in any description of the method. The implicit concept identified here is named with a term which gives the meaning of the concept (some useful

terms have been listed in the glossary as standard in the use of the framework in section 3.6), and then recorded with a parentheses in the column 'fundamental concept' in the principle part of the table.

— *Assess the relationship between 'fundamental principle' and 'fundamental concept'*

The following criteria may be used for such assessment:

(a) For each concept in the 'fundamental concept' column, which principle does it support and depend on?

— *Classify the principle 'abstraction'*

The study of object-oriented methods available shows that the principle 'abstraction' is used in almost every method but with different emphases on the aspects (i.e., data, process, or behaviour) of a system, as stated in Section 3.3.1. The types of abstraction in the method should be classified by examining the corresponding fundamental concepts and models, so that the emphasis of object modelling can be recognised and understood. The following criteria may be used in this case:

(a) Does the model include the elements (e.g., 'attribute', 'association', etc.) that are often, in particular, used to specify the static structure of objects or is there a concept that also means the data dependency of objects?

If yes, 'abstraction' in the method refers to the 'abstraction with data'.

(b) Does the model include the elements (e.g., 'operation', 'collaboration', etc.) that are often used to specify the behaviour of objects without temporal concerns or is there a concept that also means the functional dependency of objects?

If yes, 'abstraction' in the method refers to the 'abstraction with process'.

(c) Does the model include elements (e.g., 'state', 'event', etc.) that are usually used to specify the behaviour of objects over time or is there a concept that means the dynamic behaviour of objects?

If yes, 'abstraction' in the method refers to the 'abstraction with behaviour'.

The features dependent on each other are catalogued and recorded in the same row, see Table 3.6. Such a table provides an overview of the features and their dependencies, included in the 'what' aspect of the method: the vertical columns show individual features in the 'what' aspect of the method, while the horizontal rows illustrate the chain of dependency of one feature on another.

Fundamental Principle	Fundamental Concept	Model	
		Element	Type
Abstraction (data)	Object Class	Object Class	Object Model
Encapsulation Abstraction (behaviour)	Object behaviour	Attribute Operation Object state	
Inheritance	Inheritance relationship	Gen-Spec relationship	

Table 3.6 Dependency of the Essential Features in the 'What' Aspect

— Check the 'pyramid' construction of the features described in the table

By focusing on each row in the table, the following criteria may be used to check the dependencies between features:

(a) Are the definitions and meanings of the features in the same row consistent with and complete, one to another, according to the method?

(b) Does there exist any other implicit principle or concept which is fundamental to object-oriented analysis and is Refer to by an element in the models in the method?

(c) Could be this implicit principle or concept replaced by an existing explicit principle or concept in Table 3.1?

Action 2.1.2: Assess the content of the 'what' aspect

Based on the 'pyramid' construction of the features in the 'what' aspect of a method, the content of the 'what' aspect of the method is assessed so that the content of the features can be understood more accurately. The criteria as follows may be helpful to the assessment:

(a) Is there any new or extra information about the features in this aspect according to their 'pyramid' construction?

(b) What knowledge about the features can be learnt from, and derived from, the assessment of the relationship in action 2.1.1?

This action helps to further consider and understand the features in the 'what' aspect of a method. It may also show that different concepts in different methods may be defined upon the 'same' principle and the same concept may be supported by different elements in the models in different methods (See Chapter 5).

3.5.2.2 Activity 2.2: Assess the 'How' Aspect of the Method

In terms of the practice part of the framework, the five kinds of essential features (i.e., *notation, tactic of analysis, input of analysis, process of analysis*, and *product of analysis*) that reflect the 'how' aspect of an analysis method were identified in the previous stage. These features are further assessed in this activity by examining the role of each feature in the method, in order to understand how they support analysis in practice. Additionally, the dependencies and relationships between the features is also considered and assessed here, as some features may have an impact, one on another, in the method. Our study of object-oriented analysis methods makes it clear that the essential features 'notation', 'tactic of analysis', 'input of analysis' and 'products of analysis' usually have impact on the feature 'process of analysis' in a method. Because of a data-driven tactic of analysis, for example, the process of analysis in the method often focuses on identifying and specifying the static structure of objects including 'attributes' of objects and associations between objects. Whereas, with a process-driven tactic of analysis, the process of analysis usually emphasises the identification and specification of the behaviour of objects without temporal concerns, including the operations of and the collaborations between objects. Figure 3.3 illustrates the general relationships between the feature 'process of analysis' and other features in the 'how' aspect of an analysis method. The interview with the user as one potential source of input of analysis is also considered in this diagram, as the method may require the analyst to do so. The assessment of such relationships will help to understand the 'how' aspect of a method in a wider and deeper level.

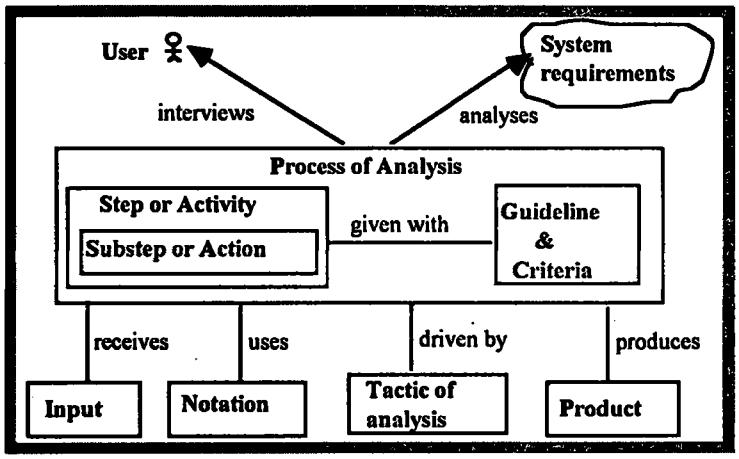

Figure 3.3 Relationships between the Essential Features in the 'How' Aspect

Action 2.2.1: Assess the relationships between the features in the 'how' aspect

— Assess the relationship between 'notation' and 'process of analysis'

The following criteria are used in the assessment of the relationship between 'notation' and 'process of analysis':

(a) Which notation is used, for each step or activity in the process?

(b) Which symbol in the notation is used, for each step/substep or activity/action in the process?

(c) Is there any instruction of using a symbol in the notation?

If some symbol is not used in any step/substep or activity/action, it is possible that the process of analysis is not complete, or that only a subset of the notation is used by this method.

— Assess the relationship between 'input' and 'process of analysis'

Different inputs have different impacts on the process of analysis. An incomplete input needs a process that enables the analyst obtain a complete input during analysis such as by a dialogue with users (e.g., the OMT method) or through prototyping a supposed system (e.g., the Booch method). It is therefore significant to understand the relationship between 'input' and 'process of analysis' in an analysis method. The assessment of the relationship between 'input' and 'process of analysis' is determined using the following criteria:

(a) which step/substep or activity/action in the process accesses the input of analysis?

(b) what kind of information is this step/substep or activity/action responsible for finding from the input and how does it do it?

(c) where does the required information in the input come from (e.g., problem statements or dialogue with users), according to the step/substep or activity/action?

If the input of analysis is implicit in the description of the method, it will gradually become explicit using the assessment in this action. The input of analysis identified here is then recorded in the corresponding column in the practice part in Table 3.1.

— Assess the relationship between 'tactic of analysis' and 'process of analysis'

To assess the relationship between 'tactic of analysis' and 'process of analysis', the following criteria may be used:

(a) Do the process of analysis and the tactic of analysis emphasise the same aspect of a system?

(b) How does this tactic impact on the process of analysis, e.g., priority of steps/substeps or activities/actions?

(c) According to the priority of steps/substeps or activities/actions in the process of analysis, which aspect of a system is emphasised? And what kind of tactic of analysis should be used to drive this process in the method?

The first two criteria are used when the tactic of analysis is known from the description of a method. Otherwise the last criterion should be used to find the implicit tactic of analysis in a method.

— Assess the relationship between 'product' and 'process of analysis'

The relationship between 'product' and 'process of analysis' is assessed by using the following criteria:

(a) What is the outcome of each step/substep or activity/action in the process of analysis?

(b) How is the outcome of each step/substep or activity/action related to the outcome of another? And how is the whole product of analysis generated step/substep by step/substep or activity/action by activity/action in the process of analysis?

(c) Does the process cover all parts of the product required according to the method?

Action 2.2.2: Assess the content of the 'how' aspect

Based on the above assessment of the features in the 'how' aspect of an analysis method, the content of the 'how' aspect of the method is assessed here so that the content of these features can be understood more accurately. The following criteria may be helpful to the assessment:

(a) Is there any new or extra information about the features in this aspect according to the relationships identified?

(b) What knowledge about these features can be learnt from, and derived from, the assessment of the relationships in action 2.2.1?

3.5.2.3 Activity 2.3: Assess the Relationships between the Two Aspects of the Method

As discussed in Section 3.2.1, the 'what' aspect of a method usually has impact on the 'how' aspect of the method as the latter aims to support the former in analysis. Methods normally carry out analysis through a process that transforms the input of analysis (e.g., an application scenario) into the product of analysis (e.g., an object model), as shown in Figure 3.4. The other essential features of the method can be illustrated around the process of analysis (see Figure 3.5), according to the roles of, and the relationships between, these features in object-oriented analysis.

Based on Figure 3.5, the relationships between the 'what' and 'how' aspects of a method are assessed by examining how the former has impact on the latter and how the latter supports the former, as illustrated in Figure 3.6. The major aim of such an assessment is to understand how the method combines these two aspects together to realise object-oriented analysis. This will be useful for people in understanding the method more widely and deeply. Figure 3.6 shows that only the feature 'model' in the 'what' aspect is directly related to the features in the 'how' aspect. The assessment of the relationships between the two aspects, therefore, is in fact the assessment of the

relationships between the feature 'model' in the 'what' aspect and the features in the 'how' aspect.

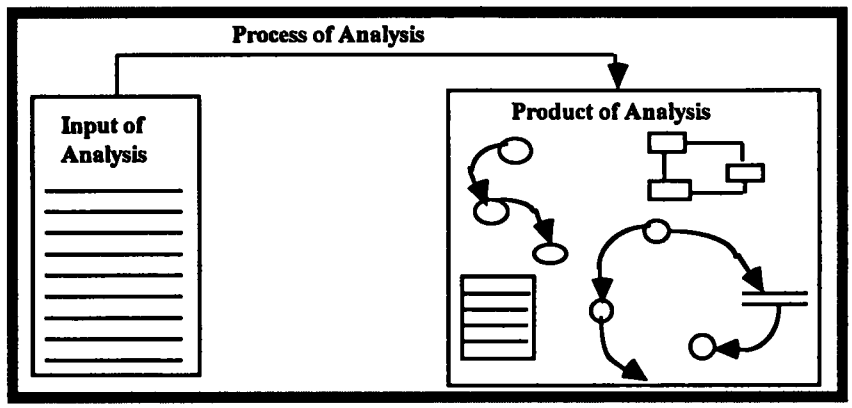

Figure 3.4 Object-Oriented Analysis

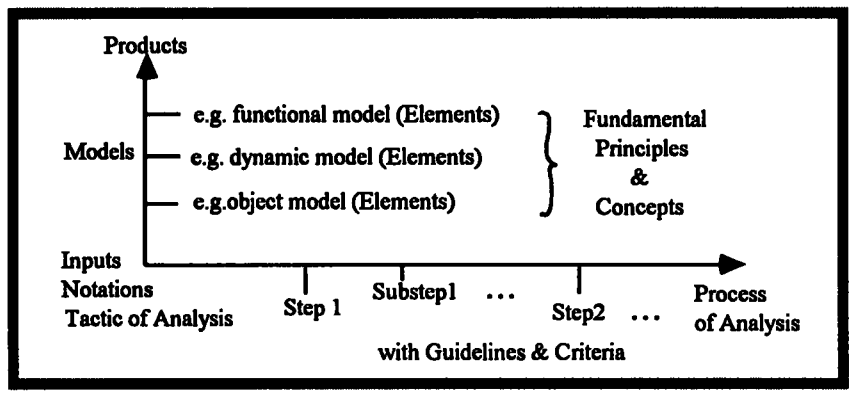

Figure 3.5 Roles of the Essential Features in Object-Oriented Analysis

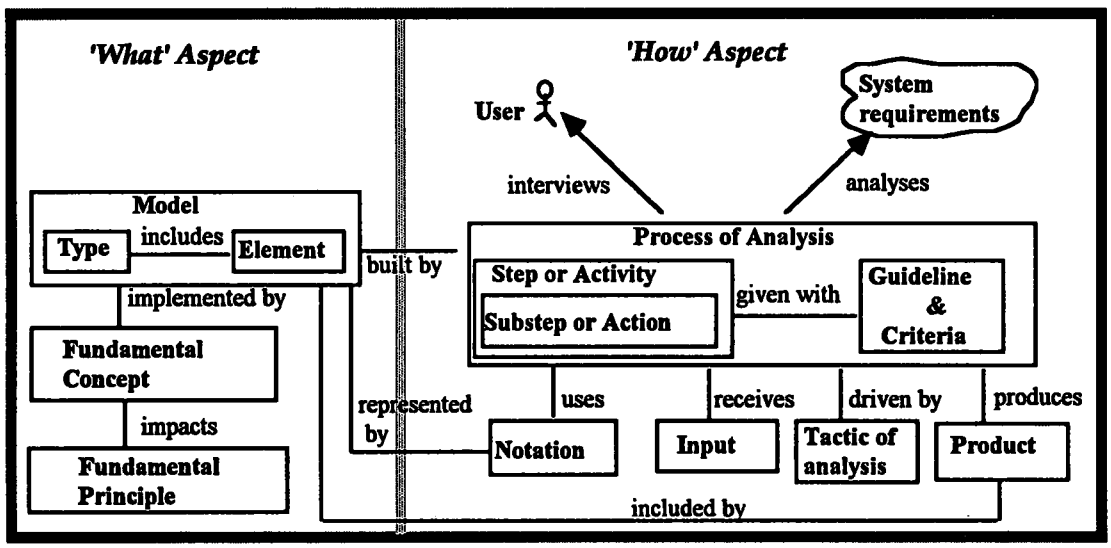

Figure 3.6 The Relationships between the 'What' Aspect and the 'How' Aspect

Action 2.3.1: Assess the relationship between 'model' and 'notation'

In an analysis method, a model of a system has to be represented by a notation that may be either graphical or textual or mixture of both. In particular, the aspect of the system emphasised by the model must be specified explicitly by the notation. That is, if a model focuses on the data aspect of a system using objects, the notation used to represent this model should include the symbols that can represent the static structure of objects for the model. For example, the OOA model [Coad91a] focuses on the data aspect of a system, so the OOA method notation includes the symbols (see Figure 4.1) which represent the static structure of objects (i.e., 'attribute', 'instance connection', 'Gen-Spec structure' and 'whole-part structure') in the system. To assess the relationship between 'model' and 'notation' finds the correspondence between a specific symbol in the notation and an element in the model. On the other hand, a model can be represented by different types of notation. The assessment of the relationship between 'model' and 'notation' also helps to select an appropriate notation for a specific model. The criteria for the assessment are as follows:

(a) Which symbol in the notation corresponds to which element in the model?

(b) How does this symbol represent that element? And is there any constraint or condition on the use of the symbol?

(c) Is there an alternative symbol in the notation that also represents the same element?

(d) Is there any element in the model which is not covered by the notation and, if so, why?

Action 2.3.2: Assess the relationship between 'model' and 'tactic of analysis'

The study of object-oriented methods shows that, in general, if an analysis method focuses on the data aspect of a system, this method usually provides special elements such as 'attribute' and 'association' in a model to specify this aspect. Corresponding to the model, the tactic of analysis in the method is often data-driven. Similarly, if the method focuses on the process aspect of the system, the above elements may not be included in

the model and the tactic of analysis now is often process-driven. The tactic of analysis therefore can be recognised by checking the elements in the models and exploring the aspect emphasised by the method, in terms of following criteria:

(a) Which aspect of a system is emphasised according to the focus of the primary model in the method?

(b) Which tactic of analysis should be provided in order to enforce the process of analysis to build the primary model first?

(c) Is this tactic same as the tactic of analysis claimed in the method?

Action 2.3.3: Assess the relationship between 'model' and 'process of analysis'

In a method, the contents of the models have strong impact on the content of the process of analysis. The assessment of the relationships between these two features is useful in understanding the dependency between the features, and additionally, the differences between the process of analysis in a method and the process of analysis in other methods. This is also helpful in recognising the primary model in the method.

— Assess the relationship between 'model' and 'step or activity'

To assess such a relationship, the following criteria may be helpful:

(a) How many steps/substeps or activities/actions are provided to build a model?

(b) Does the process of analysis cover all models which the method aims to build ?

(c) What is the priority of building models according to the process of analysis?

The model built first should be the primary model and other models are built upon it.

(d) In which step/substep or activity/action is an element in each model defined?

(e) Does the process of analysis cover every element in each model?

— Assess the relationship between 'model' and 'guideline and criterion'

The correspondence between 'model' and 'guideline and criterion' should be considered in order to understand the detail of the model and, in particular, the content of each element in the model. For example, the OMT method provides these criteria in the substep "identify objects and classes": *the significant object classes should not be*

redundant, irrelevant, vague, attributes, operations, or implementation constructs. These criteria included in the substep reveal more information about the element 'object' in the object model and make the meaning of this element more comprehensible. To assess the relationship between 'model' and 'guideline and criterion', the following criteria are used:

(a) Does the process contain the guidelines and criteria for building each model?

(b) Do the guidelines and criteria reveal more about the model and its elements than their definitions identified and described in a previous stage?

(c) Are the guidelines and criteria consistent with the meanings of the model and its elements?

(d) If they are not consistent, which of them is more appropriate and acceptable according to the context of the method?

Action 2.3.4: Assess the relationship between 'model' and 'product of analysis'

Models are the main components of the product of analysis, as delivered to design, since they specify what an object-oriented system should do according to the requirements of the system. The model therefore has impact on the product of analysis in the method. For instance, the product of analysis in the OMT method contains three types of model; that is, the object model, the dynamic model and the functional model; while the product of analysis in the Booch method only has one model; that is, the object model. The assessment of the relationships between 'model' and 'product of analysis' clarifies the content of the product of analysis in the method and gives an understanding of how the models affect the outcomes of analysis. The following criteria are useful in the assessment:

(a) Does the product of analysis contain the model?

(b) Does the product of analysis also include extra documents?

(c) What is the standard of product of analysis according to the method?

(d) Is there any constraint or condition on the product of analysis, according to the model, such as a specific standard?

3.6 Glossary

The terms defined and used in the above framework are summarised below, so that they can be easily referred to when and where they are used. Additionally the glossary includes the terms that represent the essential features, such as 'fundamental principle', 'fundamental concept' and 'tactic of analysis', that may not be explicitly defined in the text of an object-oriented method but that are implicitly used together with other features of the method. These additional terms are referred to only when a method does not define the features with a term.

Abstraction: the principle of analysing a system by focusing on some aspect of a system in analysis. This aspect may be a data aspect, a process aspect, or a behaviour aspect. (Refer to Section 3.3.1)

Abstraction with behaviour: abstraction of object behaviour within a system with time dependency. Such abstraction is concerned with the sequence of states and operations of objects over time. (Refer to Section 3.3.1)

Abstraction with data: abstraction of the static structure of objects within a system. Such abstraction is concerned with the attributes of objects and the relationships between objects. (Refer to Section 3.3.1)

Abstraction with process: abstraction of object behaviour within a system without time dependence. Such abstraction is concerned with the operations of objects and the functional collaboration between objects. (Refer to Section 3.3.1)

Aggregation: a notion of connecting an aggregate object and its part objects. (Refer to Section 3.3.2)

Criterion: a rule for selecting and specifying the correct elements for the model(s) of a system. (Refer to Section 3.4.4.3)

Communication with message: the interaction between objects by sending messages to one another. (Refer to Section 3.3.1)

Data-driven tactic of analysis: a strategy of analysing a system by focusing on the static structure of objects within the system, i.e., the attributes of objects and the static relationships between objects within a system. (Refer to Section 3.4.2)

Encapsulation: a principle of combining data and processes into objects and hiding the internal details of an object behind its external aspects. (Refer to Section 3.3.1)

Framework: a configuration of an analysis method in which the method is viewed by focusing on its 'what' aspect and its 'how' aspect (i.e., what it intends to do and how it does object-oriented analysis), and each aspect refers to a set of essential features of the method. (Refer to Section 3.2)

Fundamental concept: the general idea or notion of objects in a system underlying the models in an analysis method. (Refer to Section 3.3.2)

Fundamental principle: the basic law of object-oriented analysis underpinning the object concepts in an analysis method. (Refer to Section 3.3.1)

Grouping: a notion of collecting the objects which are tightly related to one another within a system. (Refer to Section 3.3.2)

Guideline: the detail of a step/substep or an activity/action in the process of analysis which enables the analyst to carry out the corresponding task of analysis with confidence. (Refer to Section 3.4.4.3)

Inheritance: a principle in object-orientation, behind the concept 'inheritance relationship', that means one class may share the properties and behaviour of other classes. (Refer to Section 3.3.1)

Input of analysis: the user's requirements which may be a problem domain, a dialogue between the user and the analyst, or a problem statement. (Refer to Section 3.4.3)

Object interaction: the functional dependency between two objects, i.e., one object performs its operation upon another. (Refer to Section 3.3.2)

Model: the description or specification of a system based upon the user's requirements. A model consists a set of elements each of which shows a piece of the system. (Refer to Section 3.3.3)

Notation: the symbolism used to represent a model. (Refer to Section 3.4.1)

Object behaviour: a notion of a sequence of the operations (or history) of an object over time in connection with the states of the object. (Refer to Section 3.3.2)

Object life cycle: a notion that describes the transformation of the states of an object from its creation to its deletion. (Refer to Section 3.3.2)

Object structure: a notion that describes the dependency between objects within a system. (Refer to Section 3.3.2)

Partitioning: a notion of decomposing a large system into smaller parts. (Refer to Section 3.3.2)

Process of analysis: a set of activities, or a sequence of steps, by which the input of analysis is analysed in order to build the model(s) of a system, resulting in the product of analysis. (Refer to Section 3.4.4)

Process-driven tactic of analysis: a strategy of analysing a system by focusing on the objects and their operations and collaborations within the system. (Refer to Section 3.4.2)

Product of analysis: the deliverable from analysis to design which may be a model and documents (and so on) giving the specification of a system. (Refer to Section 3.4.5)

Scale: a notion of partition of a system or a grouping of objects. (Refer to Section 3.3.1)

Subsystem: a notion of a part of a system that consists of a set of the objects which are tightly related to each other within the system. (Refer to Section 3.3.2)

System function: a notion of a collection of the operations of objects that describe what a system may do according to the requirements of a system. (Refer to Section 3.3.2)

Tactic of analysis: the strategy of analysing a system which enforces the analysis to focus on some specific aspect of the system. (Refer to Section 3.4.2)

Chapter 4

Analysis of Five Analysis Methods Using the Framework

To understand the four object-oriented analysis methods [Coad91a, Rumbaugh91, Booch91, Wirfs90] that were used in the study in Chapter 2 in an objective and systematic way, they are assessed here by using the framework defined in Chapter 3. In addition, this chapter provides examples of applications of the framework. Another (new) object-oriented analysis method, Syntropy [Cook94a], which is claimed as a second-generation object-oriented analysis method, is also assessed in terms of the framework. The results of the assessment of Syntropy will be helpful in exploring the essential features of a second-generation object-oriented analysis method. By the process provided in Section 3.5 — in particular, the criteria included in each activity/action — each method is analysed by the first stage by means of the components of the two parts of the framework, so that the explicitly essential features that reflect the 'what' and 'how' aspects of the method can be identified and described. The method is then assessed by the second stage by assessing the content of the features identified and the relationships between the features. In addition, any implicitly essential features of the method are identified and determined at this stage.

This chapter carries out the first stage of the assessment of these five analysis methods. The definition or meaning of each feature in each of the methods is shown here in detail, providing the documentation of the features for the assessment of the method in the next chapter.

4.1 Stage 1: Analyse the OOA Method (Coad and Yourdon)

The OOA method [Coad91a] focuses on object-oriented analysis but not object-oriented design. In terms of the components of the two parts in the framework of Table 3.1, the OOA method is analysed in this section and its essential features are identified focusing on the 'what' and 'how' aspects of the method.

4.1.1 Activity 1.1: Analyse the 'What' Aspect of the OOA Method

Action 1.1.1: Identify Fundamental Principles

By using the criteria of this action, the following principles are used by this method, being useful in managing the complexity of a problem domain and the system's responsibilities within it:

a) Abstraction

The OOA method considers that abstraction can help an analyst to choose certain things over others even if he/she is very familiar with a problem domain. Data abstraction is regarded as the basic abstraction in this method.

b) Encapsulation (also information hiding)

Encapsulation is a principle that the interface to each object is defined in such a way as to reveal as little as possible about its internal details.

c) Inheritance

Inheritance is a mechanism for expressing similarity among classes, simplifying the definitions of classes similar to those previously defined.

d) Association

Association is used for relating together things in object modelling that occur at the same point in time or under similar circumstance. (It seems more appropriate to treat this as a concept rather than as a principle according to the framework, since it is not a basic law of analysis.)

e) Communication with messages

This is a basic principle for interactions between objects.

f) Pervading methods of organisation

This is a principle that is used within the method to help to think about 'objects and attributes', 'whole and parts', and 'classes, members, and distinguishing between them' in analysis.

g) Scale

Scale is a principle that guides a reader through a large model by partitioning it into smaller parts.

h) Categories of behaviour

This principle focuses on three common types of behaviour of objects: event-response, change over time, and similarity of functions.

Action 1.1.2: List Fundamental Concepts

The OOA method claims that it addresses the following equation to realise object-oriented analysis:

Object-oriented = Classes and Objects + Inheritance + Communication with messages.

Four fundamental concepts are identified by this equation, according to the criteria of the action:

a) Object

An object is an abstraction of something in a problem domain, reflecting the capabilities of a system to keep information about it, interact with it, or both.

b) Class

A class is a description of one or more objects with similar properties.

c) Inheritance relationship

An inheritance relationship is a notion to represent explicitly the commonality of classes.

d) Message

A message means any communication, written or oral, sent between objects.

Action 1.1.3: Analyse Models

An object model is built by this method. By using the criteria of the action, the detail of the model is explored as follows. This model focuses on the data aspect of a

system by means of five layers: *class-&-objects layer, structure layer, attribute layer, subject layer* and *service layer.* The behaviour aspect of the system is also described by the model although it receives less emphasis than the data aspect. The elements included in the object model are as follows:

a) Class-&-objects, class

A class-&-objects is an *encapsulation* of attribute values and their exclusive services. 'Class-&-objects' means *"a class and the objects in that class"* and it is a description of one or more objects in the class. If there is no object in the class, the encapsulation is decribed by a 'class'.

b) Attribute

An attribute is some data (that is, state information) for which each object in a class has its own value.

c) Service

A service is a specific behaviour that an object is responsible for exhibiting.

d) Object state

An object state describes the current values of the attributes in an object.

e) Gen-Spec structure

A Gen-Spec (that is, generalisation-specialisation) structure represents the '*is a*' or '*is a kind of*' relationships among classes: a specialisation class is a kind of generalisation class and it inherits the definition of the latter. In addition, multiple Gen-Spec structures (that is, a lattice inheritance) are allowed by the method.

f) Whole-Part structure

A whole-part structure represents a 'has' relationship among class-&-objects: a 'whole' class-&-objects has one or more 'part' class-&-objects.

g) Instance connection

An instance connection represents a dependency between two objects one of which needs another object to help to fulfill its responsibilities.

h) Message connection

A message connection models the functional dependency of one object on another object, and indicates a need for services to fulfill its responsibilities.

i) Subject

A subject is a mechanism for simplifying a large, complex model by partitioning the model into small pieces.

As the results of this activity, the essential features identified are recorded in Table 4.1 in connection with their definitions above.

Principle / Part / Stage	Fundamental Principle	Fundamental Concept	Model	
			Type	Element
Analysis	•Abstraction (data) •Encapsulation (also information hiding) •Inheritance •Communication with messages •Pervading methods of organisation •Scale •Categories of behaviour	•Object •Class •Inheritance relationship •Message •Association	•OOA model	•Class-&-Objects •Class (i.e.without instance) •Attribute •Service •Object state •Gen-Spec structure (single, multiple) •Whole-part structure •Instance connection •Message connection •Subject

Table 4.1 The Essential Features in the 'What' Aspect of the OOA Method

4.1.2 Activity 1.2: Analyse the 'How' Aspect of the OOA Method

In this activity, the essential features that reflect the 'how' aspect of the OOA method are identified by analysing the method, in terms of the practice part of the framework.

Action 1.2.1: Illustrate the Notation

On the basis of the criteria of the action, the notation (and associated symbols) used to represent the object model and its elements are identified and illustrated in Figure 4.1. Additional notations, *Object State Diagrams* and *Service Charts*, are also identified. The symbols used in these notations are shown in Figure 4.2 and 4.3, respectively. The former

describes the states of a class-&-objects and the latter specifies an algorithm of a service in a class-&-objects.

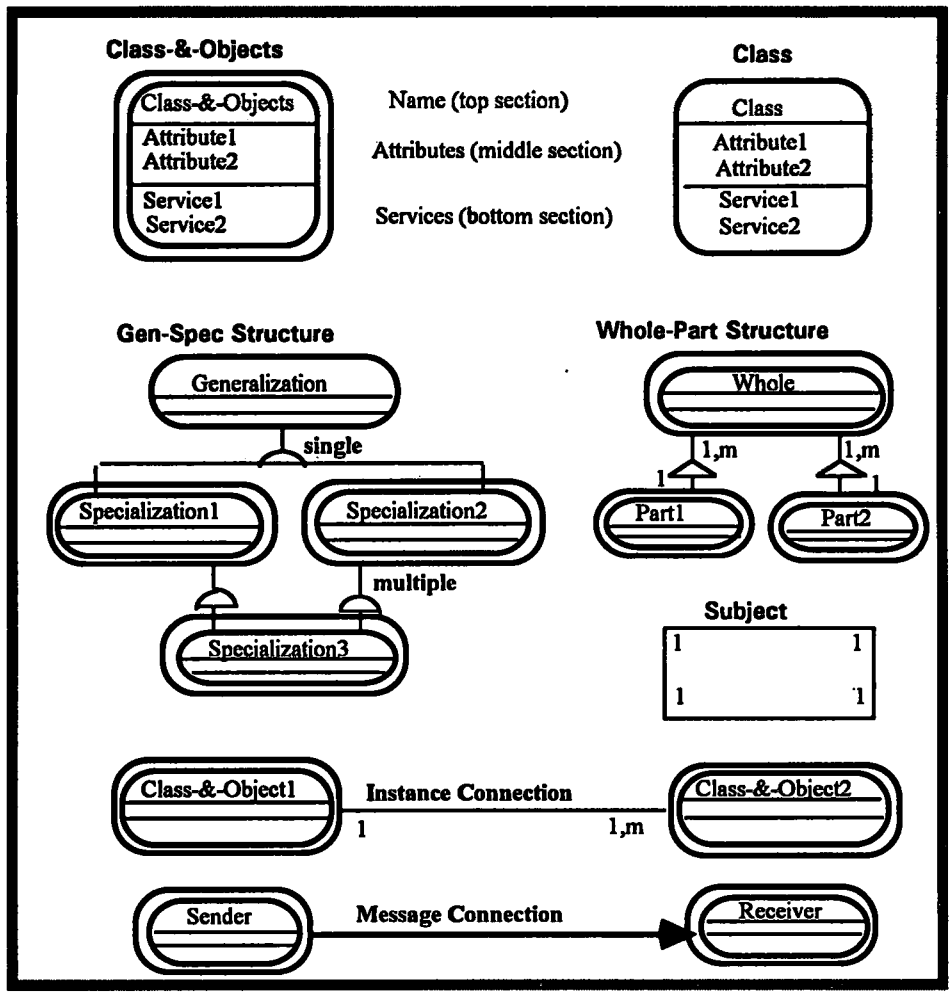

Figure 4.1 OOA Notation

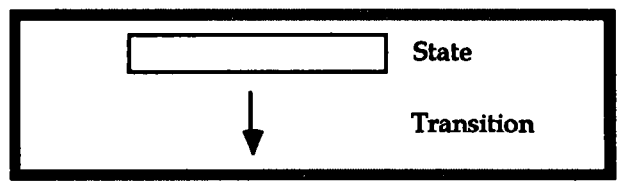

Figure 4.2 The Symbols in Object State Diagrams

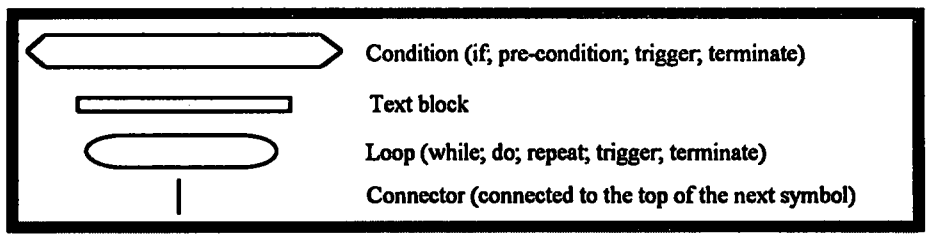

Figure 4.3 The Symbols in Service Charts

In order to describe the detail of a class-&-objects, the *class-&-objects specification template* is used as follows:

Specification
attribute
externalInput (i.e., the data from outside)
externalOutput (i.e., the data to outside)
objectStateDiagram
additionalConstraints (e.g., timing and sizing)
notes
service<name & Service Chart>
and, as needed,
traceabilityCodes
applicableStateCodes
timeRequirements
memoryRequirements.

Action 1.2.2: Identify the Tactic of Analysis

No tactic of analysis is found in the text of the OOA method, and so the identification of the tactic has to be delayed until the second stage of the assessment.

Action 1.2.3: List the Input of Analysis

The input of analysis in the OOA method, i.e., the system requirements, is any stylised problem domain which comes from the problem statements or the dialogue with users. The input is allowed to be incomplete at the beginning of analysis, according to the method.

Action 1.2.4: Describe the Process of Analysis

The process of analysis of the OOA method comprises five activities. They are activities but not steps, since no sequence is assumed upon them. Each activity is also partitioned into a collection of actions each of which performs a piece of work in the activity such as *where*, *what*, *why* and *how* a system does it. In addition, guidelines are given for each action, as well as the criteria for determining the significant elements in a model of a system.

Activity 1: Finding Class-&-Objects (for the class-&-objects layer)

— *Where to look*

Guidelines: Observe first-hand; listen actively; check previous results from the method; reuse the class-&-objects defined in other systems.

— *What to look for*

Guidelines: Identify potential class-&-objects by looking for structures, other systems, devices, things or events that 'remember', roles played, operational procedures, sites, and organisational units from the problem domain.

— *What to consider and challenge*

Criteria: Determine the significant class-&-objects on the basis of 'needed remembrance', 'needed behaviour', (usually) multiple attributes (i.e., 'If an Object has just one Attribute, get suspicious ...), (usually) more than one object in a class, always applicable attributes (otherwise, explore a Gen-Spec structure), always-applicable services, domain based requirements, and avoiding derived results.

Activity 2: Identifying Structures (for the structure layer)

— *What to look for*

Guidelines: Consider each class as a generalisation or specialisation to identify Gen-Spec structures; consider all objects with three variations—assembly-parts, container-contents, and collection-members—to identify whole-part structures.

— *What to consider and challenge*

Criteria: Determine the significant structures by checking if each is within the system's responsibilities, if there is inheritance between classes, if the specialisation is a significant class-&-objects, or if a whole-part structure provides a useful abstraction in dealing with the problem domain.

Activity 3: Identifying Subjects (for the subject layer)

— *Select subjects*

Guidelines: Promote the uppermost class in each structure upwards to a subject. Then promote each class-&-objects that is not in a structure upwards to a subject.

— *Refine subjects*

Guidelines: Subjects are refined by using problem sub-domains and considering minimal dependencies (instance collections) and minimal interactions (message connections) between the subjects.

— *Construct subject layers*

Guidelines: Group the class-&-objects into a subject. A class-&-objects may be in more than one subject. A subject may contain other subjects in a multi-level map to guide a reader through a larger model.

Activity 4: Defining Attributes (for the attribute layer)

— *Identify the attributes*

Guidelines: Look for candidate attributes for each class-&-objectsby asking the questions such as "what do I need to know?" and "what state information do I need to remember over time?". Make each attribute capture an 'atomic concept' (i.e., a single value, or a tightly-related grouping of values).

— *Position the attributes*

Guidelines: Put each attribute into the class-&-objects that it best describes, i.e., it may be necessary to apply inheritance in Gen-Spec structures: position the general attributes higher and the specialised attributes lower.

— *Identify instance connections*

Guidelines: Identify the connections between objects. Each connection shows that one object may need another in order to fulfill its responsibilities. For each object, add connection lines and define the range with each line (e.g., one-to-one, one-to-many, etc.). The following special cases should be checked: many-to-many (that is, if a new object is needed to add), connections between objects of the same class (that is, if it is significant), and those with special constraints (that is, if a new attribute should be added to an object).

Activity 5: Defining Services (for the service layer)

— *Identify object states*

Guidelines: As a service is a specific behaviour that an object may perform, it is related to some state of the object. This task identifies the states of an object through examining the potential values for the attributes and determining if the system's responsibilities include different behaviours for these values. The states and transitions in an object are described by an object state diagram.

— *Identify the required services*

Guidelines: Identify algorithmically-complex services rather than algorithmically-simple services (i.e., create, connect, access and release) that are not explicitly specified in a model. The services are identified by looking for the calculations that calculate results from the values of attributes.

— *Identify message connections*

Guidelines: Identify the message connections for each object: (1) by drawing an arrow from this object to another that needs a service from this object; and (2) by drawing an arrow from another object to this object, where the other object provides a service to this object.

— *Specify the services*

Guidelines: Specify each service in a class-&-objects by filling a *class-&-objects template* and drawing a *service chart.*

Action 1.2.5: Identify the Products of Analysis

By using the criteria in the action, the products of analysis from the OOA method are a documentation set that includes:

— a five layer OOA model,

— class-&-objects specifications, and

— other documentation, as needed.

The explicit essential features that reflect the 'how' aspect of the OOA method have been identified above. They are recorded in Table 4.2.

Practice Part / Stage	Notation	Tactic of Analysis	Input	Process of Analysis			Product
				Activity	Action	Guideline & Criterion	
Analysis	•OOA notation •Object state chart •Service chart •Class-&-objects specification template	?	•A problem domain in any style (e.g. problem statements, or dialogue)	•Finding class-&-objects	•Where to look •What to look for •What to consider and challenge	•Observe first-hand, listen actively, ... •Looking for structures, other systems,..... •Needed rememberance, needed behaviour,...	•A five-layer model •Class-&-object specifications •Other documents, if needed
				•Identifying structures	•What to look for •What to consider and challenge	•Consider each class as a generalisation,..... •Check if it is in the problem domain	
				•Identifying subjects	•Select subjects •Refine subjects •Construct subject layers	•Promote the uppermost class in each ..... •Use problem sub-domain and consider ..... •For a large model, consider using a	
				•Defining attributes	•Identify attributes •Position attributes •Identify instance connections	•What do I need to know? ..... •Apply inheritance in Gen-Spec structure,... •For each object, add connection lines	
				•Defining services	•Identify object states •Identify required services •identify message connections •Specify the services	•Examine the potential values for ..... •Look for the calculations which ..... •Draw an arrow from this object to ..... •Draw services charts •Use specification template with a	

Table 4.2 The Essential Features in the 'How' Aspect of the OOA Method

4.2 Stage 1: Analyse the OMT Method (Rumbaugh et al.)

The essential features of the OMT method [Rumbaugh91] are identified in terms of the framework by following the process of assessment of section 3.5, including the use of the criteria offered. The features show that, in general, this method does object-oriented analysis by analysing the problem statements and then constructing three kinds of model for the (object-oriented) system. The analyst is encouraged to work with the requester, since the initial problem statements are rarely complete and correct. The essential features identified are shown as follows.

4.2.1 Activity 1.1: Analyse the 'What' Aspect of the OMT Method

Action 1.1.1: Identify Fundamental Principles

By using the criteria of the section, the fundamental principles (which are claimed as the themes of object-oriented analysis) used in this method are identified below.

a) Abstraction

In the OMT method, abstraction enables an analyst to focus on the essential and inherent aspects of an entity and to ignore others.

b) Encapsulation (also information hiding)

Encapsulation is a principle which separates the external aspects of an object from the internal ones so that only the external aspects can be seen by other objects.

c) Combining Data and Behaviour

This is a principle that means an object is an entity comprising both data structure and behaviour.

d) Inheritance

Inheritance is a principle that enables classes to share similar data structure and behaviour.

e) Emphasis on Object Structure, not Procedure Structure

This method assumes that the object structure is more stable than the behaviour. This principle is thus used by this method.

Action 1.1.2: List Fundamental Concepts

In the OMT method, the term 'object-oriented' means that software is organised as a collection of discrete objects that incorporate both data structure and behaviour. Thus the following object concepts are regarded as significant by the method:

a) Object

An object describes a concept, abstraction, or thing with crisp boundaries and meanings for the problem at hand. It combines both data structure and behaviour in a single entity.

b) Identity

Identity is a concept that means the data is quantified into discrete, distinguishable objects such that an object must have its own inherent identity to avoid ambiguity.

c) Classification

Classification is a concept that means objects having the same properties ought to be grouped together.

d) Polymorphism

Polymorphism is a concept that means an operation may behave differently in different classes. (However, it is a notion concerned with the implementation of operations and so it should not be regarded as fundamental to analysis, according to the criteria of action 1.1.2.)

e) Inheritance relationship

An inheritance relationship is a notion that means that the definition of a class can be shared by another, if needed.

f) Object behaviour

Object behaviour is a concept which means that an object has its own behaviour, i.e., its states and operations performed to make state changes.

g) Object Structure

Object structure is a concept that means the properties of objects and links between objects.

Action 1.1.3: Analyse Models

The OMT method emphasises that the models built by object-oriented analysis should not contain computer (that is, implementation) constructs. Instead they should capture the structures of objects from the real world that are important to the application.

• Types of Model

It is found that three types of analysis models are provided by the OMT method: *object model, dynamic model,* and *functional model.* The object model emphasises the structure of objects in a system — objects identities, the relationships between objects, the attributes and operations of objects. It is the primary model in the OMT method. The dynamic model specifies the behaviour of objects in the object model — that is, the states of the objects and the interactions between the objects — by showing the event traces and event flows between objects. The functional model focuses on the data value transformations within the system. This model specifies the meanings of operations, as well as any constraints, in the object model and the actions in the dynamic model. Generally these models refer to each other by sharing the same objects and classes.

• Elements in Each Model

Different elements are included in the three kinds of model as follows.

— Object Model

The elements in the object model are identified as follows, by means of the criteria of this action.

a) Object

An object has attributes, state, and operations. All objects are distinguishable in terms of their identities.

b) Class

A class is a group of objects that have common attributes and operations. A class represents the objects that have a common definition. In addition, the class that has no objects is called an 'abstract' class.

c) Attribute

An attribute is a data value, not an object, held by the objects in a class. For example, 'name' and 'age' may be the attributes within a class 'Person'.

d) Operation

An operation is a function or transformation that may be applied to the objects in a class. For example, 'hire' and 'fire' may be the operations in a class 'Company'. All objects in a class share the same operations.

e) Link, association

A link is a physical or conceptual relationship between objects. An association describes a group of links with common structures and common semantics. In addition, other specific elements are also provided to describe the links and associations: *link constraints, link attributes, role names* and *qualified links.*

f) Aggregation

If two objects are tightly bound by a whole-part relationship, it is an aggregation. An aggregation object consists of the 'part' objects. However, aggregation is regarded as a special form of association in the OMT method.

g) Generalisation

A generalisation is a relationship between a class and one or more refined versions of it. The class being refined is call the *superclass,* and each refined version is called a *subclass.*

h) Constraint

A constraint represents a condition on, or a functional relationship between, components such as objects, classes, attributes, links or associations in the object model. It restricts the values that the components can assume in a system.

i) Module

A module is a logical construct for grouping classes, associations, and generalisations.

— Dynamic Model

The following elements in the dynamic model are identified by this action:

a) Event

An event is something that happens at a point in time. It may be a signal, input, decision, interrupt, transaction, or action that is an individual stimulus from one state to another within an object or from one object to another.

b) Object state

An object state is an abstraction of the attribute values and links held by an object. The values in a state affect the behaviour of an object. This state specifies the response of the object to input events, and it is often associated with the value of an object satisfying some condition.

c) Activity

An activity is associated with a state and it is an operation that takes time to complete.

d) Action

An action is associated with an event and it is an instantaneous operation that is performed in response to the corresponding state or event.

— Functional Model

The following is a list of the elements in the functional model:

a) Process

A process is an operation that transforms data values.

b) Data Flow

A data flow connects the output of an actor or process to the input of another actor or process.

c) Actor

An actor is an active object that produces or consumes values.

d) Data Store

A data store is a passive object that stores data for later access.

All the explicit features identified above reflect the 'what' aspect of the OMT method. They are recorded in Table 4.3.

Principle / Part / Stage	Fundamental Principle	Fundamental Concept	Model	
			Type	Element
Analysis	•Abstraction •Encapsulation (also information hiding) •Combining data and behaviour •Inheritance •Emphasis on object structure not procedure structure	•Object •Identity •Classification •Inheritance relationship •Object behaviour •Object structure	•Object Model	•Object •Class •Abstract class •Attribute •Operation •Association, link •Constraint •Aggregation •Generalisation (single, multiple) •Module
			•Dynamic Model	•Event •Object state •Activity •Action
			•Functional Model	•Process •Data flow •Actor •Data store

Table 4.3 The Essential Features in the 'What' Aspect of the OMT Method

4.2.2 Activity 1.2: Analyse the 'How' Aspect of the OMT method

Action 1.2.1: Illustrate the Notation

By means of the criteria of this action, three kinds of notation are used in this method: *object diagrams, state diagrams* and *data flow diagrams*, representing the object model, the dynamic model and the functional model, respectively. Additionally, *event trace diagrams* and *event flow diagrams* are used to assist in the creation of state diagrams.

a) Object Diagram

Object diagrams are a graphic notation that describe objects, classes and their relationships. The symbols included in this notation are shown in Figure 4.4

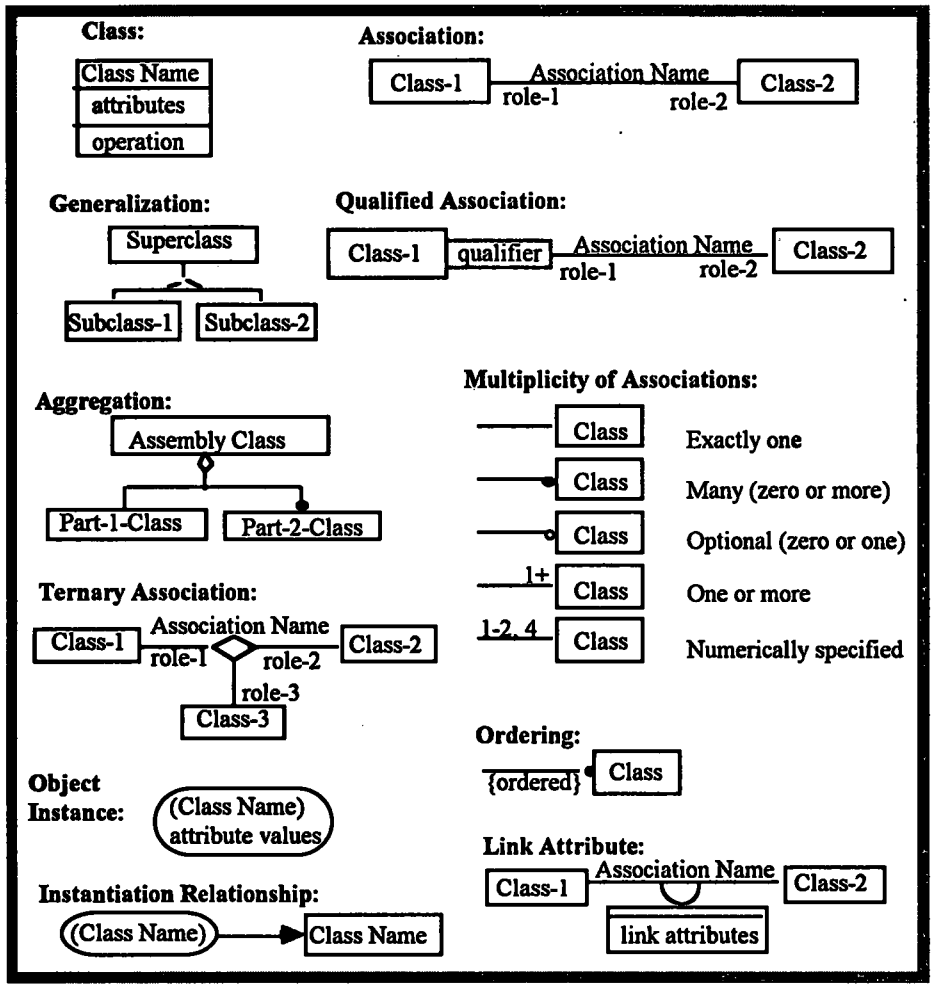

Figure 4.4 The Symbols in Object Diagrams

b) State Diagrams

A state diagram is used to describe the behaviour of a single class of objects that have the same behaviour and share the same class features. The symbols included in state diagrams are shown in Figure 4.5.

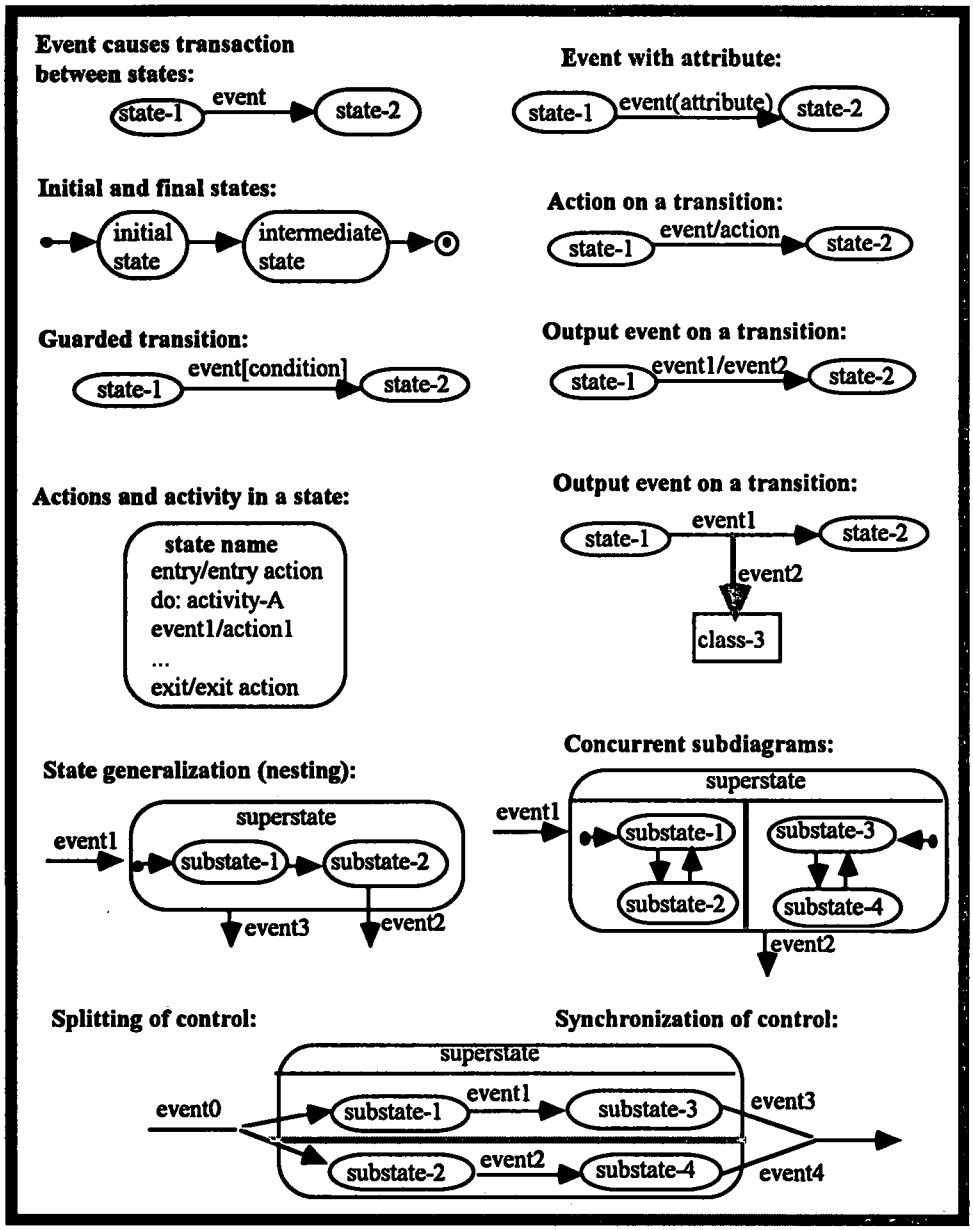

Figure 4.5 The Symbols in State Diagrams

c) Data Flow Diagrams

A data flow diagram describes the transformations from inputs to outputs within a system. The symbols included in data flow diagrams are given in Figure 4.6.

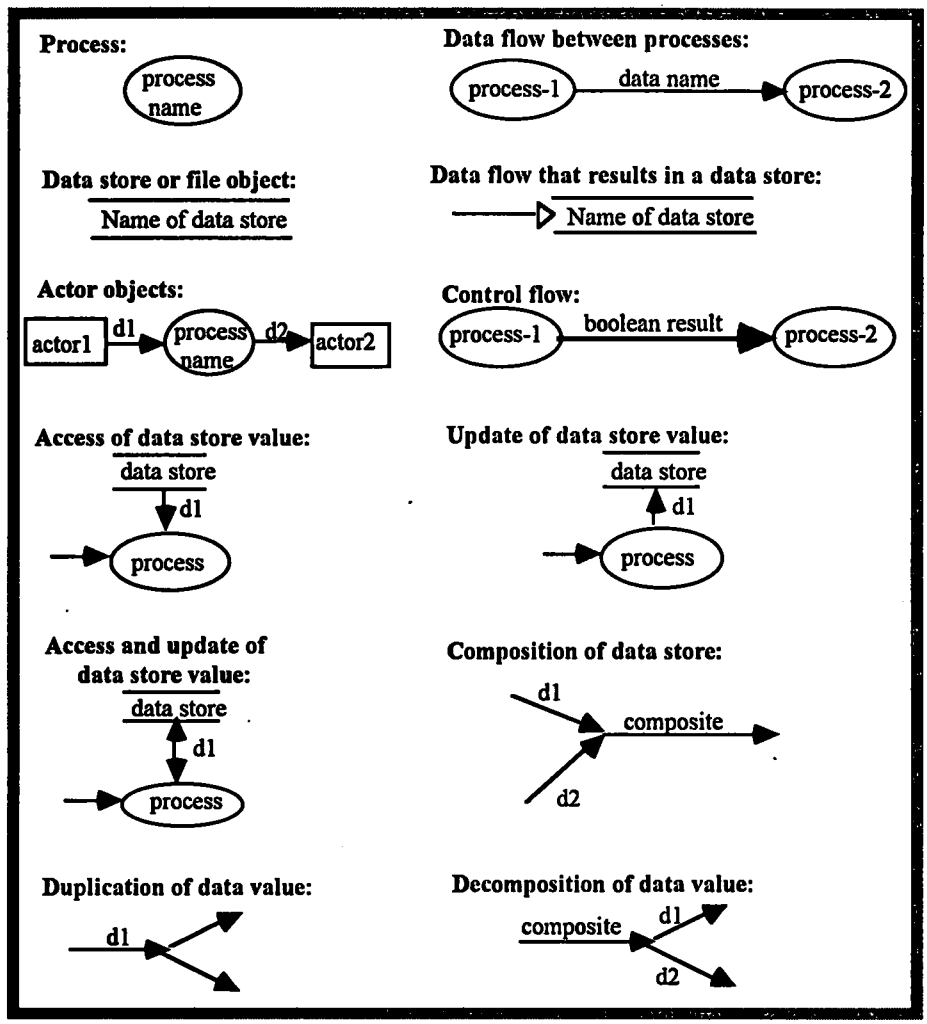

Figure 4.6 The Symbols in Data Flow Diagrams

Action 1.2.2: Identify the Tactic of Analysis

No tactic of analysis is found in the text of the OMT method, similar to the situation with the OOA method above. The tactic cannot be decided until the next stage of assessment, i.e., the assessment of the relationship between 'model' and 'tactic of analysis'.

Action 1.2.3: List the Input of Analysis

The initial input of the analysis required by the OMT method is found to be the problem statements. The analysis is started even if such an input is not yet complete. A dialogue with users may be needed since the experience of software development shows that initial statements are rarely complete and correct.

Action 1.2.4: Describe the Process of Analysis

The process of the analysis in this method is identified and described below, by means of the criteria of section 1.2.4. This process consists of five steps. Each step also includes a sequence of substeps together with the guidelines and criteria for carrying out each substep and for determining the significant components in each model.

Step 1: Constructing the Object Model

— *Identify objects and classes*

Guidelines: Identify relevant objects and classes from the problem statements by considering the nouns in the statements that describe physical entities and concepts in the problem domain.

Criteria: The significant objects and classes should not be redundant, irrelevant, vague, attributes, operations, or implementation constructs.

— *Prepare a data dictionary*

Guidelines: In order to describe precisely each object and class, create a data dictionary for containing definitions of all objects and classes.

— *Identify associations between objects*

Guidelines: Identify the associations between objects by considering the verbs and verb phrases in the problem statements.

Criteria: A significant association should not be an association between eliminated classes, an irrelevant or implementation association, an action or event, a ternary association, or a derived association.

— *Identify attributes of objects and links*

Guidelines: Identify attributes by considering the nouns followed by possessive phrases, or adjectives in the problem statements.

Criteria: An attribute of an object should not be an object, a qualifier, a name which depends on the context, an identifier, a link attribute that depends on the presence of a link, an attribute of which the values are invisible externally, an attribute that is unlikely

to affect most operations, and a discordant attribute that seems completely different from and unrelated to all other attributes in the class.

— *Identify inheritance between classes*

Guidelines: Organise the classes that share common structures by using inheritance relationships: to generalise common aspects of existing classes into a superclass (bottom up) or to refine existing classes into specialised subclasses (top down); in an inheritance structure, attributes and associations should be assigned to the most general class that it is appropriate to.

— *Test access paths*

Guidelines: To find the missing information, such as constraints on the use of the data or an object in the object model, the access paths in the model is tested according to the problem statements.

— *Iterate and refine the model*

Guidelines: Iterate and refine the model by checking if there are missing objects or unnecessary object classes, missing associations or unnecessary associations, or incorrect placement of associations or attributes.

—*Group classes into modules*

Guidelines: Group the objects and classes that describe a logical subset of the object model into a module.

Step 2: Constructing the Dynamic Model

— *Prepare a scenario of typical interaction sequences*

Guidelines: First prepare a scenario for the 'normal' cases (i.e., the interactions without any unusual inputs or error conditions); then consider the 'special' cases (e.g., omitted input sequences, maximum and minimum values, or repeated values); finally consider user error cases (e.g., invalid values and failures to respond).

— *Identify events between objects*

Guidelines: Identify all external events by examining the scenario: all signals, inputs, decisions, interrupts, transitions, and actions to or from users or external devices. Then

allocate each event to the objects and classes that send it and receive it. Draw an event flow diagram to show possible control flows between objects.

— *Draw a state diagram for each object*

Guidelines: The behaviour of each object is described here by a state diagram in which events are received or sent by the object. However, not all objects need to be described by state diagrams. If an object receives and sends events without any further state transition within it, it is not described by a state diagram.

— *Match events between objects to verify consistency*

Criteria: Check the completeness and consistency of the dynamic model according to these criteria: every event should have a sender and a receiver; states without predecessors or successors should represent starting or terminating points of the interaction sequence; events through the system should match the scenarios; corresponding events on different state diagrams should be consistent; and synchronisation errors where an input occurs at an awkward time should be prevented since objects are inherently concurrent.

Step 3: Constructing the Functional Model

— *Identify input and output values*

Guidelines: Begin by listing input and output values that are parameters of events between a system and the outside world, and then find the input or output values missed by analysing the problem statements.

— *Draw data flow diagrams showing functional dependencies*

Guidelines: Show how each output value is produced from the input values by drawing a data flow diagram using functional decomposition.

— *Describe functions*

Guidelines: Each function is specified in an appropriate form such as a natural language that focuses on what the function does rather than how it achieves it.

— *Identify constrains between objects*

Guidelines: Here constrains mean the functional dependencies, but not the input-output dependencies between objects.

— *Specify optimisation criteria*

Guidelines: Give the criteria to optimise the function such as maximised or minimised values.

Step 4: Adding Operations

— *Operations from the object model*

Guidelines: The OMT method assumes that attribute values must be accessible to operations. These operations however may not to be shown explicitly in the object model. Instead they may be implied by the attributes.

— *Operations from events*

Guidelines: During analysis, events should not be explicitly listed in the object model and they are best represented as labels on state transitions in the dynamic model. The operations corresponding to events are implied by the events.

— *Operations from state actions and activities*

Guidelines: In the dynamic model, actions and activities may correspond to operations if they have significant computational structures. These operations should be put in the object model.

— *Operation from functions*

Guidelines: In the functional model, each process that has significant computational structure corresponds to an operation within one or more objects. These operations should be put in the object model.

— *Consider shopping list operations*

Guidelines: Sometimes the operations that are not dependent on a particular application but meaningful to the future possible needs are implied by the real-word behaviour of objects and classes. These operations are called 'shopping list' operations. They may be put in the object model, if necessary.

— *Simplify operations*

Guidelines: The operations that are put in the object model should be simplified under these guidelines: use inheritance where possible to reduce the number of distinct operations; introduce new superclasses as needed to simplify the operations; locate each operation at the correct level in the class hierarchy.

Step 5: Iterating the Analysis

— *Refine the analysis model*

Guidelines: Refine object definitions to increase sharing and improve structure; add details that were glossed over during the first analysis.

— *Restate the requirements*

Guidelines: Check and confirm the system requirements and verify the models against the requirements.

Action 1.2.5: Identify the Products of Analysis

It is found that the products of the analysis generated by the OMT method for a system are as follows:

— A data dictionary that defines (using natural language) every class within a system,

— An object model that shows the objects, classes, their attributes and operations, and their relationships in a system,

— A dynamic model that shows the behaviours of objects in the system, and

— A functional model that shows the transformations from input to output in the system.

The above are the description of the essential features that reflect the 'how' aspect of the OMT method. As a graphical representation, they are recorded in Table 4.4.

Practice Part / Stage	Notation	Tactic of Analysis	Input	Process of Analysis			Product
				Step	Substep	Guideline & Criterion	
Analysis	•Object Diagram •State Diagram •Data Flow Diagram	?	•A problem statements •dialogue with experts	•Constructing the object model	•Identify objects and classes •Prepare a data dictionary •Identify associations •Identify attributes •Identify inheritance •Test access paths •Iterate and refine the model •Group object classes into modules	•Consider the nouns in the statements..... •Contain all objects and classes •Condier the verbs and verb phrases in ..... •Consider the nouns followed by ..... •Generalise common aspects of existing ... •Test paths according to the problem ..... •Sign of missing associations, signs of ..... •Group a logical subset of the object model...	•A data dictionary •An object model •A dynamic model •A functional model
				•Constructing the dynamic model	•Prepare a scenario •Identify events between objects •Draw a state diagram •Match events for verification	•First prepare the scenario for "normal case".. •Examine the scenario: all signals, input, ... •If an object receives and sends events ..... •Every event should have a sender and a	
				•Constructing the functional model	•Identify input and output values •Build data flow diagrams •Describe functions •Identify constraints •Specify optimisation criteria •Operations from the object model	•Begin by listing input and output values ... •Working forward or tracing forward ..... •Each function is described in an ..... •Preconditions can be described as the ..... •The values need to be maximised, ... •The operations for reading and writing	
				•Adding operations	•Operations from events, actions and activities •Operations from functions •Conder shopping list operations •Simplify operations	•Events should not be explicitly listed in the object model and they are best represented ... •Put the processes as operations in the object.. •Consider the operations that may be needed .. •Use inheritance where possible to reduce ...	
				•Iterating the analysis	•Refine the analysis model •Restate the requirements	•To refine object definitions to increase ... •Check and confirm the requirements and ...	

Table 4.4 The Essential Features in the 'How' Aspect of the OMT Method

4.3 Stage 1: Analyse the Booch Method

By following the process of assessment, the essential features that reflect the 'what' and 'how' aspects of the Booch method [Booch91] are identified as follows, in terms of the framework.

4.3.1 Activity 1.1: Analyse the 'What' Aspect of the Booch Method

Action 1.1.1: Identify Fundamental Principles

Four principles are identified from the Booch method by means of the criteria of this action:

a) Abstraction

Abstraction is a principle to denote the essential characteristics of an object that distinguish it from other objects.

b) Encapsulation (also information hiding)

Encapsulation is a principle concerned with hiding all of the implementation details of an object that do not contribute to the essential characteristics of the object. Encapsulation hides the internal view of the object.

c) System decomposition

System decomposition is a principle that decomposes a system into a set of cohesive and loose subsystems. In accordance with this principle, objects and classes are packaged into subsystems in a way that makes their reuse convenient.

d) Hierarchy

Hierarchy is used by the Booch method to rank or order objects and classes by considering the sharing or structuring of the objects and classes. The principle provides a rule to organise the objects and classes in a system.

Action 1.1.2: List Fundamental Concepts

It is found that the following concepts are regarded as fundamental in the Booch method:

a) Object

An object is an abstraction of something that is an individual, identifiable item, or entity, either real or abstract, with a well-defined role in the problem domain.

b) Class

Class is the concept that captures the structure and behaviour common to all related objects.

c) Object structure

Object structure is a notion used to clarify the collaborations between objects by means of the external view of the objects.

d) Class structure

Class structure is a notion used to highlight the objects that share the same behaviour in a system.

e) Subsystem

Subsystem is a notion used to describe a building block for the physical structure of a system.

Action 1.1.3: Analyse Models

Only one object model is built by the Booch method, specifying the logical aspect of a system. This model describes both data and behaviour aspects of the system. The following elements are found in this model:

a) Object

An object has *state*, *behaviour* and *identity*. An identity is the name of the object that distinguishes the object from all other objects.

b) Class

A class is a set of similar objects that share the same structure and behaviour. An object in a class is an instance of the class. A class may not have any instances and such a class is termed an *abstract class*.

c) Object state

An object state encompasses all of the (usually static) properties of the object plus the current (usually dynamic) values of each of these properties.

d) Field

A field of an object is a repository for part of the state of an object. It is either persistent data within the object or a reference to another object. The referred object sends the value to the referring object and it must appear in the interface of the referring object as part of message passing. Thus the fields of an object constitute its structure.

e) Operation (also message)

An operation is an action in an object by which the object reacts to other objects.

f) Object relationship

A relationship between two objects simply means that the objects can send messages to one another.

g) Using relationships

A using relationship is a relationship that refers to the external view of an object and class: an object and class can use another via an interface (i.e., the used class must be visible to any clients). This relationship describes the client/server contract between classes.

h) Inheritance relationship

An inheritance relationship is a relationship among classes that share a similar structure or behaviour. This relationship may be single or multiple, i.e., one class only has one superclass or has more than one superclass.

i) Instantiation relationship

An instantiation relationship implies a process of producing a new class from a generic class. (It is an element which describes the implementation of generic classes, so it should not be used in analysis, according to the criteria of section 1.1.3.)

j) Module

A module is a container in which the logically related objects and classes are collected.

The explicit features that are identified above and that reflect the 'what' aspect of the Booch method are now recorded in Table 4.5.

Principle / Part / Stage	Fundamental Principle	Fundamental Concept	Model	
			Type	Element
Analysis	•Abstraction •Encapsulation (also information hiding) •Hierarchy (with inheritance & structuring) •System decomposition	•Object •Class •Object structure •Class structure •Subsystem	•Object Model	•Object •Class •Abstract class •Object state •Field •Operation (also message) •Object relationship •Using, Inheritance relationships •Module

Table 4.5 The Essential Features in the 'What' Aspect of the Booch Method

4.3.2 Activity 1.2: Analyse the 'How' Aspect of the Booch Method

Action 1.2.1: Illustrate the Notation

Two kinds of diagrams, *class diagrams* and *object diagrams*, are used to describe the logical aspect of a system, e.g., objects and classes, their states, and their relationships. The symbols used in the diagrams are shown in Figure 4.7 and 4.8. In addition, *state transition diagrams* (see the symbols in Figure 4.9) are used to specify object behaviour, i.e., the interactions between objects, and describe the events occurring over time. It is found that some symbols in the diagrams may not be necessary strictly for analysis as the notations are used for both analysis and design in the Booch method and some of the symbols are provided to describe the implementation detail of a system, such as the symbol 'parameter' in object diagrams.

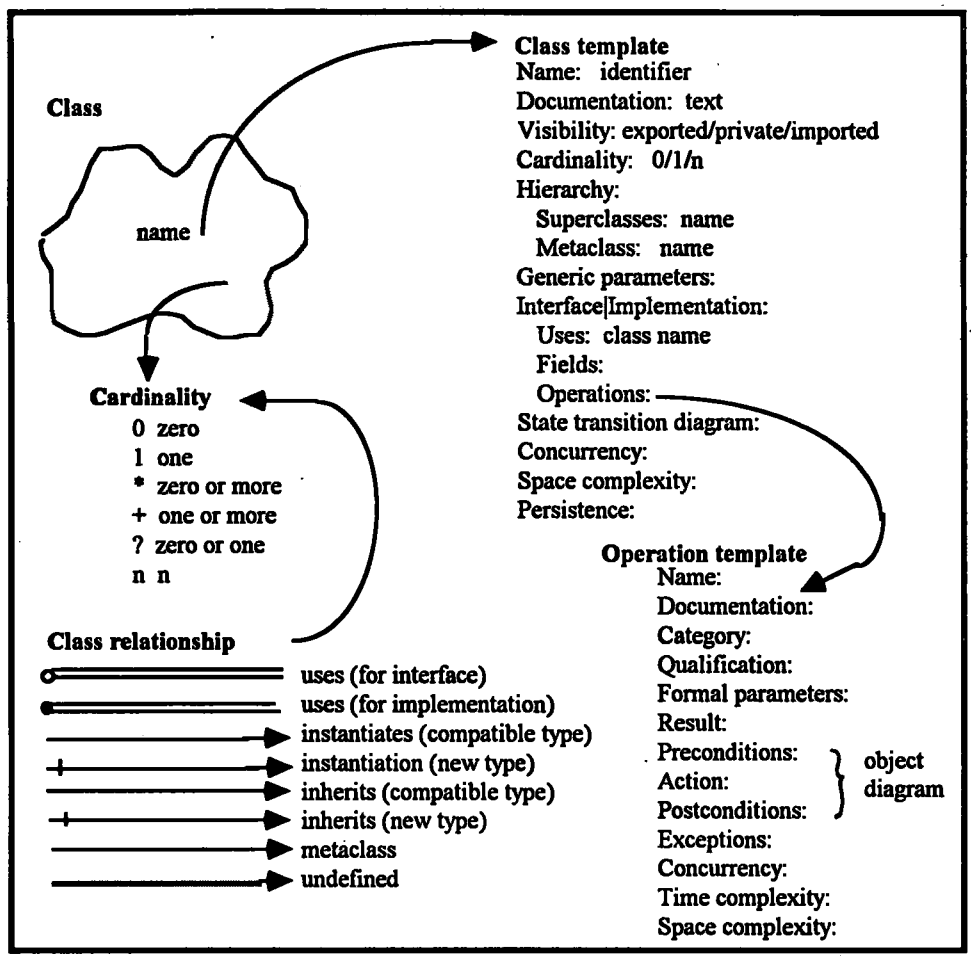

Figure 4.7 The Symbols in Class Diagrams

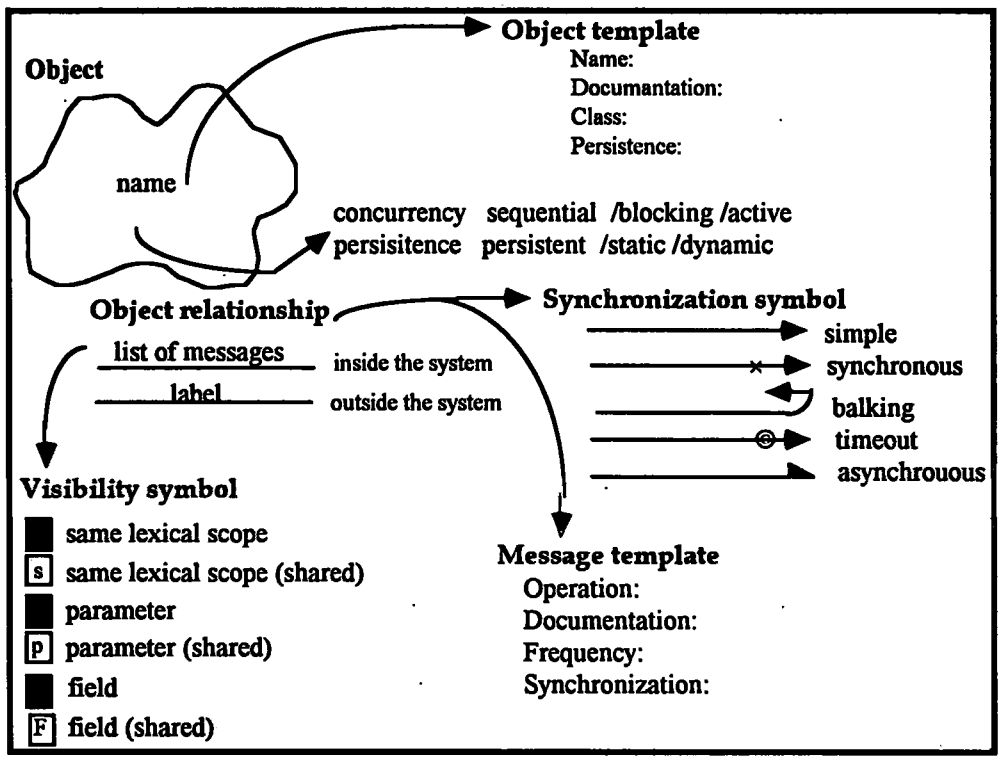

Figure 4.8 The Symbols in Object Diagrams

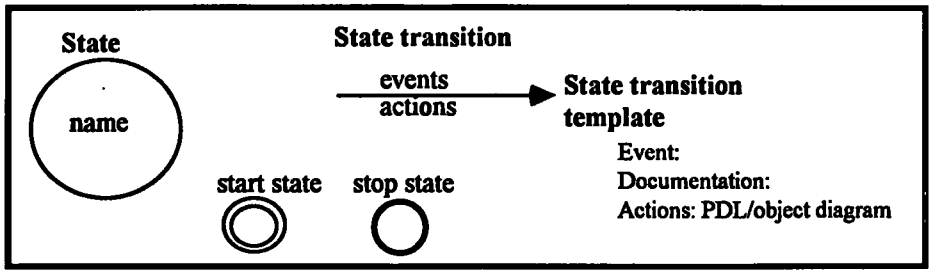

Figure 4.9 The Symbols in State Transition Diagrams

Action 1.2.2: Identify the Tactic of Analysis

No tactic of analysis is claimed in the text of this method, a situation similar to that for the OOA method and the OMT method, as described above. The tactic supported by this method, however, will be decided in action 2.3.1 in the next chapter, in the assessment of the relationship between 'model' and 'tactic of analysis'.

Action 1.2.3: List the Input of Analysis

A problem domain is the initial input of analysis for the Booch method. During analysis, an updated object model is also used as another input to the analysis when iterating.

Action 1.2.4: Describe the Process of Analysis

Booch states in his book [Booch91] that analysis in his method is part of the process of a round-trip gestalt design which discovers and defines the objects and classes in a problem domain. According to the criteria of action 1.2.4, this process is found to consist of four steps. The detail of the process and steps are described as follows.

Step 1: Identify the Classes and Objects

Guidelines: To discover essential classes and objects from the vocabulary of the problem domain by using another object-oriented analysis method or a domain analysis method. The classification approaches may be as follows:

a) *classical categorisation* that abstracts objects according to the similar properties among objects;

b) *conceptual clustering* that emphasises concepts more than properties: abstract a concept first and then decide which category it belongs to; and

c) *prototype theory*, i.e., if a class is not clearly bounded with properties or concepts, it is abstracted as a prototypical object.

Step 2: Identifying the Semantics of Classes and Objects

Guidelines: Identifies the behaviour and operations of the classes and objects. In this step, analysts need to identify the things that each class and object can do for themselves and to others. This can be done by writing a script for each object to define its life cycle and including its characteristic behaviour.

Step 3: Identifying the Relationships among Classes and Objects

— *Discover the relationships between classes and objects*

Guidelines: Identify the using and inheritance relationships between classes and the collaborations between objects.

— *Decide the visibilities between classes and objects*

Guidelines: Decide how objects or classes see each other from externally. This decision is helpful to package the classes and objects into modules.

Step 4: Implementing Classes and Objects

This step makes design decisions on objects and classes in the object model that is built by the above steps, allocates them into modules, and implements them, and so, it should not be included in the process of analysis.

Action 1.2.5: Identify the Products of Analysis

The product of analysis produced by the Booch method is an object model that shows what an object-oriented system needs to do, according to a problem domain.

The essential features identified above that reflect the 'how' aspect of the Booch method are recorded in Table 4.6.

Practice / Part / Stage	Notation	Tactic of Analysis	Input	Process of Analysis			Product
				Step	Substep	Guideline & Criterion	
Analysis	•Class Diagram •Object Diagram •State Transition Diagram	?	•A problem domain	•Identifying classes and objects		•Analyse the vocabulary of the problem domain by following one of three classification approches	•An object model
				•Identifying the semantics of classes and objects		•Writing a script for each object to define its life cycle, including its characteristic behaviour	
				•Identifying the relationships among classes and objects	•Discover the relationships between the classes and objects •Decide the visibilities between classes and objects		

Table 4.6 The Essential Features in the 'How' Aspect of the Booch Method

4.4 Stage 1: Analyse the Wirfs-Brock Method (Wirfs-Brock et al.)

The account of the Wirfs-Brock method [Wirfs90] first describes the principles behind the method, followed by a definition of the concepts fundamental to the method. The modelling elements are then provided and defined, and finally, the process of object-oriented analysis is described. The essential features that reflect the 'what' and 'how' aspects of this method are identified and described below by means of the process of assessment in section 3.5.

4.4.1 Activity 1.1: Analyse the 'What' Aspect of the Wirfs-Brock Method

Action 1.1.1: Identify Fundamental Principles

The following principles are explicitly claimed in the text of the Wirfs-Brock method [Wirfs90]:

a) Abstraction

Abstraction is regarded as the key to good software design, since it emphasises some aspect of a system and ignores others. It makes the specification of the system simple as at the abstract level the components in the system can be specified without concern for implementation details.

b) Encapsulation

This is a principle concerned with the combination in objects of the data and the operations that affect data.

c) Information hiding

This is a principle that hides the internal detail of an object from other objects.

d) Inheritance

Inheritance is a principle supporting the sharing of similarities among classes. It is regarded as fundamental by the Wirfs-Brock method since it provides a powerful way to produce reusable classes.

e) Message-sending

Message-sending makes it possible for one object to access another by sending it a message that includes its name and arguments.

f) Polymorphism

Polymorphism is a principle that allows two or more objects to respond to the same message, each in its own way. (According to the criteria of action 1.1.1, this principle focuses on design rather than analysis since it indicates how the objects are implemented in a system. It should not be regarded as fundamental in analysis.)

Action 1.1.2: List Fundamental Concepts

By using the criteria in the action, the concepts regarded as fundamental in the method are as follows.

a) Object

An object refers to a conceptual entity in the real world.

b) Class

A class is a generic specification for a set of objects that share the same behaviour.

c) Object responsibility

The object responsibility concept is a notion which specifies what an object maintains and performs in an object-oriented system.

d) Inheritance relationship

The inheritance relationship concept addresses the sharing of related classes: a new class is defined by abstracting out the similarities of existing classes (i.e., it is a superclass of existing classes), or by inheriting the behaviour from another class with something extra (i.e., it is a subclass of that class).

e) Client-server

The client-server concept is used to model the interactions between objects by considering one object (client) requests a service of another (server).

f) Subsystem

The subsystem concept is used to describe a group of classes that closely work together in order to simplify the patterns of communication between objects and to streamline the flow of control and information in a system.

Action 1.1.3: Analyse Models

Although the Wirfs-Brock method does not use a term like 'object model', the following elements are found to be provided for modelling an object-oriented system. An object model containing all of these elements can be said to be effectively built by the method.

a) Class

A class is a collection of objects that share the same behaviour. An object is the encapsulation of functions and data, and it has a public interface and a private representation to make the internal details invisible to other objects.

b) Responsibility

A responsibility describes some knowledge that an object maintains and an action that the object performs.

c) Inheritance hierarchy

An inheritance hierarchy represents the inheritance relationships between the classes in a hierarchical structure. In a particular case, a superclass may not have any object. This kind of superclass is called an *abstract class.*

d) Collaboration

A collaboration represents a request from a client class to a server class by sending a message to the server, in order to fulfill a client responsibility.

e) Client-server contract

A client-server contract represents an interaction between a client class and a server class. This contract only specifies what is needed to done rather then how it is done.

f) Subsystem

A subsystem is a group of classes, or a group of classes and other subsystems, that collaborate with one another.

The essential features, as described above, that reflect the 'what' aspect of the Wirfs-Brock method are recorded in Table 4.7.

Principle / Part / Stage	Fundamental Principle	Fundamental Concept	Model	
			Type	Element
Analysis	•Abstraction •Encapsulation •Information hiding •Inheritance •Message-sending	•Object •Class •Object responsibility •Inheritance relationship •Client-server •Subsystem	(•Object model)	•Class •Abstract class •Responsibility •Inheritance hierarchy •Collaboration •Client-server contract •Subsystem

Table 4.7 The Essential Features in the 'What' Aspect of the Wirfs-Brock Method

4.4.2 Activity 1.2: Analyse the 'How' Aspect of the Wirfs-Brock Method

Action 1.2.1: Illustrate the Notation

The following notations are found to be used in the Wirfs-Brock method:

a) Class Card and Subsystem Card

Class cards and subsystem cards describe the information about classes and subsystems, respectively. The formats of the cards are shown in Figure 4.10.

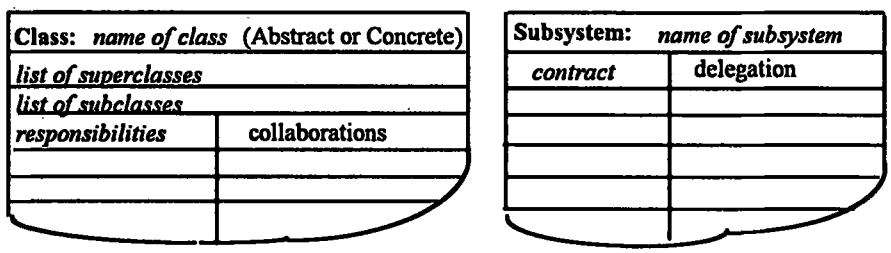

Figure 4.10 A Class Card and a Subsystem Card

b) Hierarchy Graph

This shows the inheritance relationships between classes, as shown in Figure 4.11.

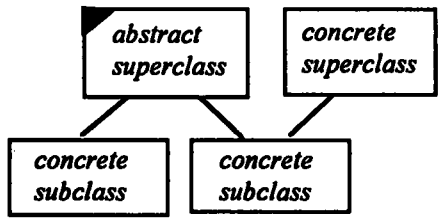

Figure 4.11 A Hierarchy Graph

c) Collaboration Graph and Contract Card

A collaboration graph, as shown in Figure 4.12, describes the classes and subsystems within a system and the paths from one to another.

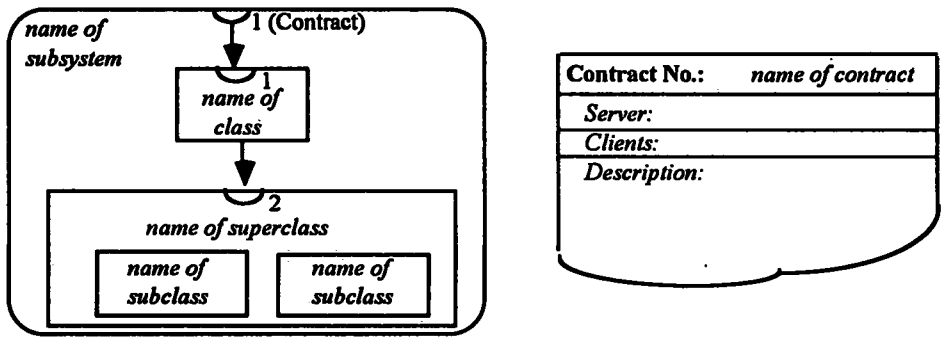

Figure 4.12 A Collaboration Graph and a Contract Card

d)*Venn Diagram*

A Venn diagram shows the common responsibilities between classes (as shown in Figure 4.13) and indicates where abstract superclasses should be created.

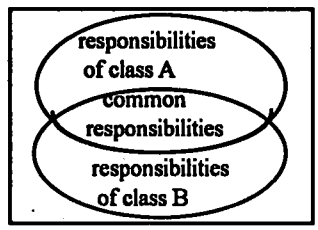

Figure 4.13 A Venn Diagram

Action 1.2.2: Identify the Tactic of Analysis

A responsibility-driven tactic is claimed by the Wirfs-Brock method. That is, a class is defined by focusing on its responsibilities and an inheritance relationship is defined by considering the common responsibilities between classes.

Action 1.2.3: List the Input of Analysis

A complete natural language specification of the system requirements is required as the input of analysis. Thus, no new input is assumed during analysis.

Action 1.2.4: Describe the Process of Analysis

There are five steps in the process of analysis for this method. Each step also consists of a series of substeps as follows.

Step 1: Finding Classes

— *Looking for noun phrases*

Guideline: All noun phases are listed by extracting them from the specification of the system requirements.

— *Choosing meaningful candidate classes*

Criteria: Meaningful classes should model the domain of the application; they should be physical objects or conceptual entities; one word for one concept; if the meaning of adjectives for the same noun imply different objects, then they are defined as different classes; if the missing subject for a sentence has the potential to be an object, a new class may be defined; if a noun phrase seems external to the system, do not define it as a class; and values of attributes may be defined as classes.

— *Finding missing classes*

Guidelines: Identify missing classes which are not explicitly described in the specification of the system. For example, an abstract superclass can be identified by grouping related classes. In addition, if similar behaviour is shared by several classes, an abstract superclass can be defined for these classes.

Step 2: Defining Responsibilities

— *Finding responsibilities from the purpose of each class*

Guidelines: The role of a class in the system is the responsibilities that it should have.

— *Extracting responsibilities from the specification*

Guidelines: The specification of a system usually gives the actions and information that a class must perform and maintain. The actions are identified by considering the verb phrases in the specification and are stated as generally as possible; keep behaviour with related information; keep information about one thing in one place; and split shared responsibilities among related objects into several smaller and more specific responsibilities and assign them separately to the most appropriate classes. If a class has no responsibility, it will be nearly always be discarded later.

— *Identifying responsibilities from the relationships between classes*

Guidelines: The relationships between classes often imply the responsibilities of the classes. In particular, the consideration of three relationships, *'is-kind-of'*, *'is-analogous-to'* and *'is-part-of'*, is useful to identify the responsibilities of classes.

Step 3: Defining Collaborations

— *Identifying collaborations by responsibilities*

Guidelines: In order to identify collaborations, questions such as these can be asked: Is the class capable to fulfilling this responsibility itself? If not, what does it need? From what other class can it acquire what it needs? In addition, each responsibility shared by classes also implies a collaboration between the classes.

— *Identifying collaboration by classes*

Guidelines: For each class, such questions as the following are asked in order to identify its responsibilities: What does this class not know? What other classes need some result or information from this class? This class should be discarded if no interaction exists between it and other classes.

— *Identifying collaborations by relationships*

Guidelines: Three relationships are useful to identify collaborations: a) 'is-part-of', such as the phrase 'are composed of' in a specification; b) 'has-knowledge-upon', such as the phrase 'which it gets from' in a specification, and c) 'depends-upon', such as the phrase

'change with' in a specification. A collaboration graph will be developed later after the contracts between classes are defined.

Step 4: Defining Hierarchies

— *Building good hierarchies*

Guidelines: In order to define each class as an abstract class or a concrete class, Venn diagrams are drawn to represent the responsibilities shared between classes. Then class hierarchies and contracts are constructed by following the guidelines: to model a 'kind-of' hierarchy; to put common responsibilities as high as possible; and to make sure that abstract classes do not inherit from concrete classes.

— *Identifying contracts*

Guidelines: The contracts are identified by following guidelines such as the following, in order to determine which responsibilities belong to which contracts: to group responsibilities used by the same clients, i.e., they all belong to one contract; to maximise the cohesiveness of classes; and to minimise the number of contracts. The fewer contracts exist, the more comprehensible a system becomes.

Step 5: Defining Subsystems

— *Defining subsystems*

Guidelines: Subsystems can help to simplify a large system. They can be identified by firstly drawing a collaboration graph for the system that shows all collaborations among classes.

Criteria: Determine the significant subsystems by asking: What is the purpose of that web? Does it show that these classes work together to implement a unit of functionality? Does it make sense to abstract the group of classes out as a single entity? Can a subsystem be built in order to subsume these classes? In addition, another way to decide whether a group is a subsystem or not is to name it. If it can be named, it is likely to be a subsystem.

— *Writing a design specification for each class and subsystem*

Guidelines: The specification for each class and subsystem is recorded by a class card and a subsystem card separately: *redraw* the graphs on pages, one page per graph; *number* the pages so that they can be referred to; and *order* the collaboration graphs from the most global to the most specific.

— *Writing a design specification for each contract*

Guidelines: The contracts between classes and subsystems are recorded by contract cards.

Action 1.2.5: Identify the Products of Analysis

The products of analysis in the Wirfs-Brock method contain the following:

— Class, subsystem and contract cards that record each class, subsystem and contract,

— Hierarchy graphs and Venn diagrams that show the inheritance relationships between classes (both abstract and concrete classes), and

— Collaboration graphs that show the contract links among the classes within a subsystem or a system.

The essential features identified and described above reflect the 'how' aspect of the Wirfs-Brock method. They are recorded in Table 4.8.

Practice / Part / Stage	Notation	Tactic of Analysis	Input	Process of Analysis			Product
				Step	Substep	Guideline & Criterion	
Analysis	•Class Card •Subsystem Card •Contract Card •Hirarchy Graph •Venn Diagram •Collaboration Graph	Analyse a complete specification of system requirements by •Responsibility -driven	•A complete specification of system requirements	•Finding classes	•Looking for noun phases •Choosing meaningful candiate classes •Finding missing classes	•All noun phrases are listed by..... •They should be physical objects,..... •Consider the classes sharing	•Object model
				•Defining responsbilities	•Finding them from the purposes of each class •Extracting them from the specification •Identifying them from the relationships	•Consider the role of it in the system. •Consider the verb phrases in the ..... •Consider three relationships: is a.....	
				•Defining collaborations	•Identifying them from responsibilities •Identifying them from classes •Identifying them from relationships	•Is the class capable of fulfilling this ..... •What does this class do not know?..... •Consider three relationships: is a.....	
				•Defining hierarchies	•Building good hierarchies •Identify contracts	•To model a 'kind of' hierarchy ..... •To group responsibilities used by ...	
				•Defining Subsystems	•Defining subsystems •Writing a specification for each class,subsystem, and contract	•What is the purposes of that web ... •Recode a class in a class card, ...	

Table 4.8 The Essential Features in the 'How' Aspect of the Wirfs-Brock Method

4.5 Stage 1: Analyse the Syntropy Method (Cook and Daniels)

The Syntropy method [Cook94a] is an object-oriented analysis and design method developed by S. Cook and J. Daniels of Object Designers Ltd. in the United Kingdom. The authors claim in their book that this method is a second-generation object-oriented method and that it is defined upon the first-generation object-oriented methods, in particular, the OMT method [Rumbaugh91] and the Booch method [Booch91], but that Syntropy gives them a more formal interpretation. The method adopts the mathematical notation given in [Hayes87] and the basic notations of Z [Abria80] for a formal description of a system. The Syntropy method is analysed and its essential features are shown in this section by viewing the 'what' and 'how' aspects of this method in terms of the framework.

4.5.1 Activity 1.1: Analyse the 'What' Aspect of the Syntropy Method

Action 1.1.1: Identify Fundamental Principles

The following principles are regarded as fundamental in the Syntropy method:

a) Abstraction

In the Syntropy method, abstraction refers to the process of focusing on and understanding the essential and inherent facts in a situation in the world, in the context of objects. A situation is a set of things and occurrences which describes some kind of activity in the world.

b) Encapsulation (also information hiding)

Encapsulation is a principle of hiding the internal detail of an object from other objects.

c) Inheritance

Inheritance is a principle that enables classes to share a description.

d) Domain

Domain means a way of dividing a system description, not of system execution, into smaller parts, i.e., subsystem descriptions.

Action 1.1.2: List Fundamental Concepts

One stated goal of the Syntropy method is to use the concepts of object technology to describe situations in the world [Cook94a]. For the purposes of object-oriented analysis, the authors consider the real world to be usefully seen to consist of components such as objects, values and events. The following fundamental concepts identified from the method are found to reflect these things in the world.

a) Object

An object is an abstraction of concrete or abstract things in a situation in the world. The properties of an object may change over time.

b) Object type

An object type is a collection of objects with the same description. It is the same as the concept 'object class' of the OMT method.

c) Identity

Identity is a concept that means an object can always be distinguished from another object.

d) Value

Value is a concept that represents a problem-domain concept like 'number' or 'string' in a situation in the world. Values are constant and they do not change in a situation.

e) Object structure

Object structure is a concept used to define the static description of a system in terms of object types and their relationships. In such a structure, a set of object types may together constitute a coherent sub-system which can meaningfully be considered as a group. Dependency between two sub-systems is a consequence of relationships between types in the sub-systems. Typical causes of dependency are visible associations and sub-type/super-type relationships.

f) Sub-typing

Sub-typing is a concept that implies object conformance: an object conforming to the sub-type also always conforms to the super-type. The description of sub-type inherits some or all of the description of the super-type, possibly with additions or modifications.

g) Object behaviour

Object behaviour is a concept to define the life history of objects in terms of events, object states and activities.

h) Event broadcast

The Syntropy method uses two basic concepts to model the world: objects and events. Objects represent things and events describe occurrences (that is, happenings or episodes rather than object occurrences). The occurrences represented by events imply also the changes of state of the things in a situation. The event broadcast concept considers that the events in modelling may be detected simultaneously by different objects.

Action 1.1.3: Analyse Models

The Syntropy method focuses on modelling aspects of a system (e.g., objects, values and sequences of events in a situation in the world) by building two sequential models: one is called an *essential* model and the other is called a *specification* model.

• Type of Model

— Essential Model and Specification Model

An essential model describes the things and concepts in a situation in the world. The purpose of an essential model is to understand a situation, real or imaginary. The building-blocks of an essential model are objects and events, and its interpretation is as a set of facts.

The specification model states what the software will do, according to the essential model, without concern for implementation details. The model specifies the states which the software can be in and the way that the software responds to stimuli (events) by

changing state and by generating responses (also events). The specification model is a refinement of the essential model. It is built in terms of events and states, like the essential model. However, it differs from the essential model. In particular, it can generate events itself because of software needs and can leave the response to an event undefined in the essential model.

— Models with Type View and State View

In this method, a situation in the world is regarded as a set of things and occurrences; that is, it consists of objects, values and events. Each of the essential and specification models are represented by two distinct views: a static view (also type view) and a state view (also dynamic view). The static view models the static structure in the situation; while the state view models the behaviour in the same situation. These two views are consistently interrelated, and the whole thing comprises a multi-dimensional model.

• Elements in the Models

The essential model and specification model consist of the same elements which are used to describe a system in both type view and state view, as shown below.

— Type view

a) Object and object type

An object consists of property values and responsibilities. An object type is a description of the property values and responsibilities that a collection of objects share. Such objects are said to be instances of the object type. Syntropy's object type element is analogous to the element 'object class' in the OMT method.

b) Property and association

A property is something that can be observed of an object in some way. For each property, an object always has its own value. A property of one objects may be another object and such a property is an association between these two objects.

c) Value type

A value type is a constraint on properties, i.e., property type. The usual syntax is:

propertyName: propertyType.

The value types used in the Syntropy method are *Number, Integer, String, Date, Time* and *Symbol.*

d) Association

An association represents a possible link between objects.

e) Aggregation

Aggregation is often referred to as a 'whole-part' or 'is-part-of' relationship, where the whole, the aggregate, is made up of its parts. In this method, aggregation implies life-time dependency, i.e., the life-time of the 'parts' is contained within the life-time of the 'whole'. The 'parts' are permanently attached to the 'whole', and cannot be removed from it without being destroyed. Conversely, destroying the 'whole' destroys the 'parts'. However, unless we can come up with some concrete semantics for whole-part relationships which go beyond those defined for associations, this element has no place in the modelling discipline in the Syntropy method.

f) Type extension (i.e., inheritance of type)

A type extension is a sub-type of another type (i.e., super-type). This is often called an 'is-kind-of' relationship. A sub-type 'inherits' all the properties, constraints and associations of its super-type. Broadly, the sub-type can extend the capabilities of the super-type but not restrict them.

g) Abstract type

An object type is an abstract type if it does not have any instance.

h) Constraints and Invariant

A constraint represents a condition on or a functional relationship between components such as objects, object types, properties or links in the essential model. An invariant may be a logical expression that will be always true for every object conforming to the type, a simple restriction on the range of property values, or a specific constraint on a property whose values remain fixed during the lifetime of its owning object.

i) Navigation expression

A navigation expression is an expression that includes a navigation through the essential model, and is a logical type constraint that will be always be true for every object conforming to the type.

j) Domain

A domain is a set of object types that together constitute a coherent sub-system which can meaningfully be considered as a group.

k) Domain dependency

Domain dependency between two domains is a consequence of relationships between types in the two domain.

— *State view*

a) Object state and state type

An object state is an abstraction of the property values and links held by an object. In principle, every different set of property values taken by an object represents a different state. The values in a state affect the behaviour of an object type. A state type is a specific sub-type of another that describes a state of its super-type.

b) State invariant

A state invariant is a condition on a state that always prevails when the object is in that state.

c) Event

An event is something that causes a situation to change from one state to another. Events are not objects but they may be operations. An object can change its state in response to any event; sometimes several objects may change their state in response to a single event. Events have no duration: either they have not yet happened or they have already happened; they can never be in the process of happening. Every event carries some information. An event may have parameters that can be object types and value types, pre-conditions that must hold for the event to occur, and sequence that lists the changes resulted from itself.

d) Event scenario

An event scenario is a sequence of specific event instances; it shows just one of the many possible sequences of events that could occur in the duration from creation to destruction of objects.

e) Creation operation

Object are dynamically created and destroyed during the life-time of a situation. A creation operation must be defined as an operation in the behaviour of objects.

f) Generation

A generation is an action that is a result from an event and attached to the transition from one state to another.

The essential features above reflect the 'what' aspect of the Syntropy method, according to the framework. They are recorded in Table 4.9.

Principle / Part / Stage	Fundamental Principle	Fundamental Concept	Model	
			Type	Element
Analysis	•Abstraction •Encapsulation (also information hiding) •Inheritance •Domain	•Object •Object tpye •Identity •Event •Value •Object structure •Sub-typing •Object behaviour •Event broadcast	•Essential Model •Specification Model	—Type view •Object •Object type •Abstract type •Property •Value type •Association •Aggregation •Type extension (single, multiple) •Constraint •Invariant •Navigation expression •Domain •Domain dependency —State view •Object state •State type •State invariant •Event •Event scenario •Creation operation •Generation

Table 4.9 The Essential Features in the 'What' Aspect of the Syntropy Method

4.5.2 Activity 1.2: Analyse the 'How' Aspect of the Syntropy Method

Action 1.2.1: Illustrate the Notation

It is claimed that the Syntropy method introduces the minimum of new notations and it adopts existing notations to represent the essential model and specification model, for instance, *OMT notation* to represent the type view of a system, Harel's *Statechart* [Harel87] to describe the state view of the same system, and the mathematics notation given in [Hayes87] to describe the constraints and invariants in both essential model and specification model. The notations used in this method are shown in Figure 4.14-16.

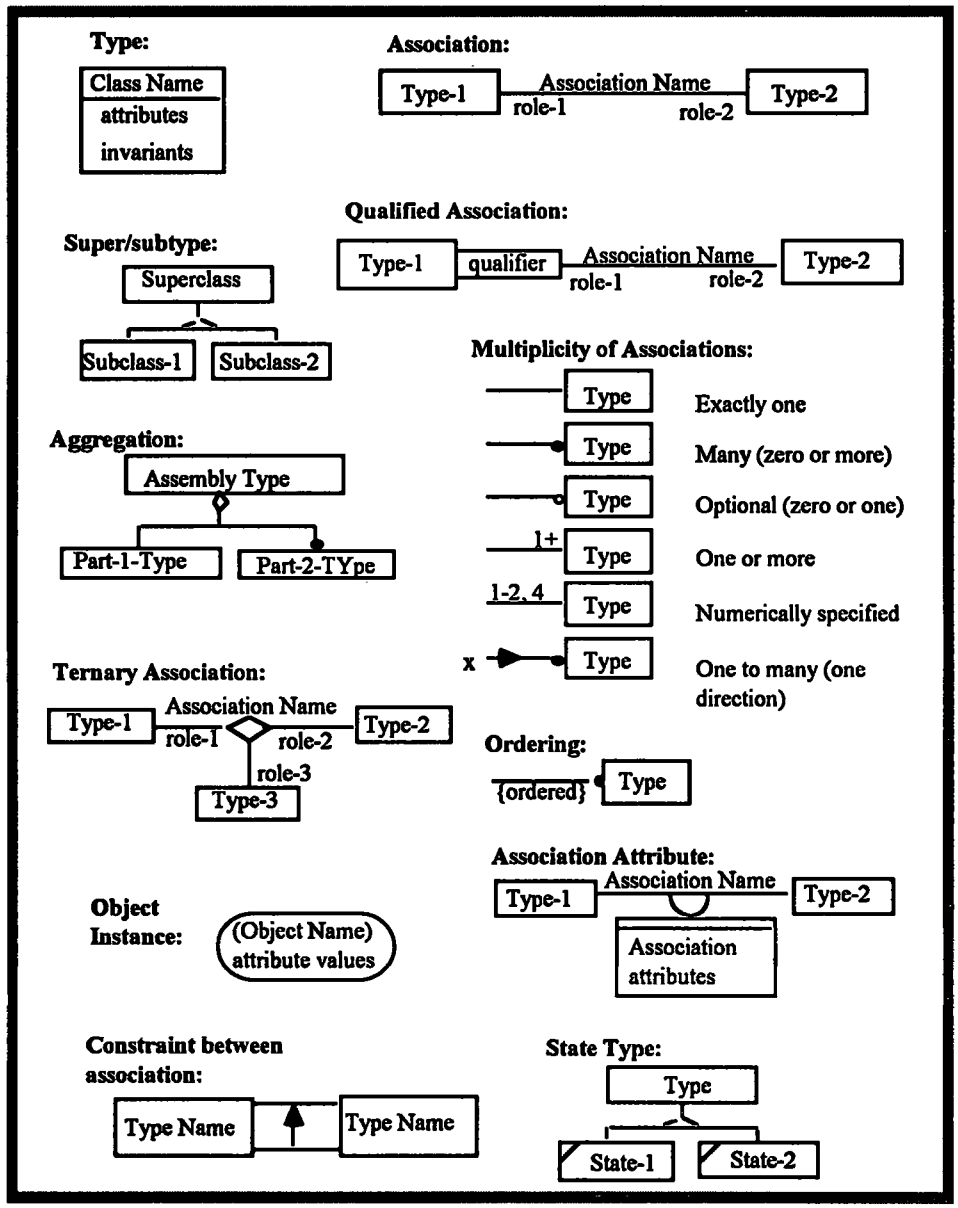

Figure 4.14 The Type View: OMT Notation

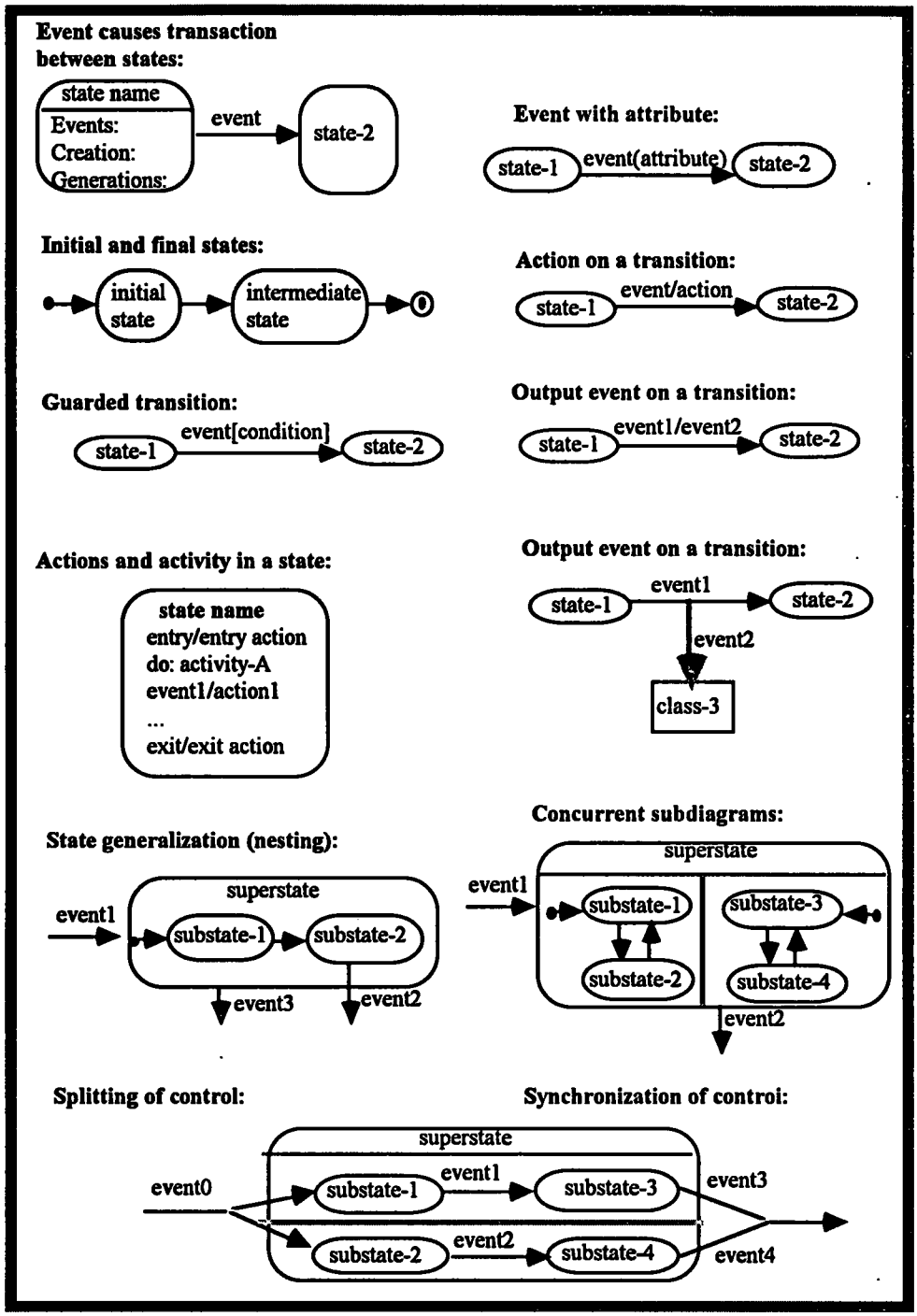

Figure 4.15 The State View: Statecharts

Notation of logic, sets and other mathematics:

— Definitions and declarations

	Meaning
LHS ≡ RHS	Definition of LHS as syntactically equivalent to RHS.
x : T	Declaration of identifier x to stand for a member of the set T (which may be a type name or any expression yielding a set).

x, y : T	≡ x : T, y : T
()	Groups terms in expressions

— Logic

	Meaning
true, false	Logical constants
not P	Negation: 'not P'
$P \wedge Q$	Conjection: 'P and Q'
$P \vee Q$	Disjunction: 'P or Q'
$P \Rightarrow Q$	Implication: 'P implies Q' or 'if P then Q'
$P \Leftrightarrow Q$	Equivalence: 'P is logically equivalent to Q' or 'P if and only if Q'
$P \rightarrow Q, R$	Conditional: 'if P then Q else R'
	$(P \rightarrow Q, R) \Leftrightarrow ((P \Rightarrow Q) \wedge (\text{not } P \Rightarrow R))$
$\forall x: S \bullet P$	Universal quantification: 'for all x in set S, P holds'.
$\exists x: S \bullet P$	Existential quantification: 'there exists a x in S such that P holds'.
$\exists! x : S \bullet P$	Unique existence: 'there exists a unique x in S such that P holds'.
$t_1 = t_2$	Equality between terms
$t_1 \neq t_2$	≡ not $(t_1 = t_2)$

— Sets

	Meaning
$t \in S$	Set membership: 't is a member of S'.
$t \notin S$	≡ not $(t \in S)$
$T \supseteq S$	Set inclusion: 'every member of S is also in T'.
{ }	The empty set.
$\{ t_1, t_2, ..., t_n\}$	The set containing the terms t_1 through t_n
#S	Size of the set S
set of S	Powerset: set of all subsets of S.
{ x : S | P}	The set containing exactly those x in S for which P holds.
{ D | P•t}	Given declarations D, the set of t's for which P holds.
{ D • t}	Given declarations D, the set of t's.
	≡ { D | true • t}
$(t_1 , t_2 ..., t_n)$	Ordered tuple of t_1, t_2 ..., and t_n
$S \cup T$	Set union.
S - T	Set difference.
$S \cap T$	Set intersection
$\cup$ SS	Distributed set union. Given SS is a set of sets with members taken from S, 'the union of all the members of all the members of all the sets'
	≡{x: S | (∃ s: SS • x ∈ s) }

$S \times T$	Cartesian product: The set of all 2-tuples such that the first component is a member of S and the second a member of T
sum S	The numerical sum all of the elements of the set S. sum { } = 0. sum ({t} $\cup$ S) = t + sum S. Also defined over sequences and bags.
min S	Minimum of a set (or sequence or bag).
max S	Maximum of a set (or sequence or bag).

— *Functions*

	Meaning
$S \rightarrow T$	The set of total functions from S to T

— *Bags*

Mathematically, a bag is treated as a function mapping elements of the bag to positive integers, representing the number of times the element appears in the bag.

	Meaning
bag of T	The set of bags whose elements are drawn from set T.
#X	The number of elements in bag X
[ ]	The empty bag
$[x_1, x_2 ..., x_n]$	The bag containing x_1, x_2 ..., x_n with the frequency in which they occur in the list.
members X	The set formed from the elements of bag X.

— *Sequences*

Mathematically, a sequence is treated as a function mapping positive integers, representing position in the sequence, to elements of the sequence.

	Meaning
seq of T	The set of sequences whose elements are drawn from set T.
#A	The length of sequence A
[ ]	The empty sequence
$[a_1 , a_2 ..., a_n]$	The sequence containing a_1 , a_2 ..., and a_n
A ^ B	The sequence formed by concatenating the sequence A with the sequence B.
A(n)	The nth element of sequence A.
members A	The set formed from the elements of A.
items A	The bag of items contained in the sequence A.
head A	The first element of a sequence or nil if the sequence is empty.

	$A \neq [\,] \rightarrow A(1)$, nil
last A	The last element of a sequence or nil if the sequence is empty.
	$A \neq [\,] \rightarrow A(\#A)$, nil
tail A	All but the head of a sequence.
front A	All but the last of a sequence.

— *Sorted sequences*

	Meaning
$S \rightarrow e$	The sorted sequence formed by inserting element e into the sorted sequence S, following the sort rule for S.

— *Objects*

	Meaning
a in Q	True if the object a is in state Q, false otherwise.

Figure 4.16 The Mathematics Notation for Constraints and Invariants

Action 1.2.2: Identify the Tactic of Analysis

The Syntropy method suggests that a situation in the real world can be analysed by starting by either identifying object types or identifying events in the situation. In the first case, the type view is first used in order to identify objects and values and then events in the same situation are identified and modelled upon the objects and values. The type view is a basis for applying the state view. A data-driven tactic of analysis is utilised by the Syntropy method in this case. In the second case, the state view is used first and the type view is then used. Once a list of events is identified, the parameters to each event must be established and the information which the event must carry is identified. Such information leads directly to the identification of the important object types in a situation in the world. An event-driven tactic of analysis is addressed in the second case by this method. Therefore there are two alternative tactics of analysis which are addressed by the Syntropy method in analysis.

Action 1.2.3: List the Input of Analysis

According to the criteria in action 1.2.3, the input to the Syntropy method is the problem statement that describes a situation in the real world.

Action 1.2.4: Describe the Process of Analysis

It can be seen in the text of the method that the process of analysis in this method should include two steps by which an essential model and a specification model are built. Each of the steps also consists of two substeps, in respect with the type view and the state view of a system. However, no further detail of the process of analysis is shown in the text of the method [Cook94a]. The identification of the process however can be delayed to the next chapter of the assessment, when the relationship between 'model' and 'process of analysis' is examined.

Action 1.2.5: Identify the Products of Analysis

The products of analysis by using the Syntropy method are identified as below:

— an essential model, and

— a specification model.

The essential features identified above reflect the 'how' aspect of the Syntropy method. They are recorded in Table 4.10.

Practice Part / Stage	Notation	Tactic of Analysis	Input	Process of Analysis			Product
				Step	Substep	Guideline & Criterion	
Analysis	•OMT notation (object diagram) •Statechart •Mathematics notation	Analyse a problem statement by •Data-driven or •Event-driven	•A problem statement	?	?	?	•An essential model •A Specification model

Table 4.10 The Essential Features in the 'How' Aspect of the Syntropy Method

Chapter 5

Assessment of Five Analysis Methods Using the Framework

The features, in particular the explicit features, that reflect the 'what' and 'how' aspects of five object-oriented methods [Coad91a, Rumbaugh91, Booch91, Wirfs90, Cook94a] were identified and described in the previous chapter, by using the framework defined in Chapter 3. They are also recorded in a tabular form as given in Table 3.1. Based upon the features identified in Chapter 4, this chapter assesses these methods according to the process of assessment provided in section 3.5. The meanings and roles of the essential features as well as the relationships between them are taken into account in the assessment, in order to clarify and understand these methods precisely and correctly. In addition, the implicit features of the methods may be recognised through this assessment. The assessment of each of the methods is shown separately in Section 5.1 to 5.5. Following this assessment, the methods are further discussed in Section 5.6, as a general review of the nature of analysis methods.

5.1 Stage 2: Assess the OOA Method (Coad and Yourdon)

This assessment stage is based on the essential features of the OOA method that were identified during the first stage in section 4.1.

5.1.1 Activity 2.1: Assess the 'What' Aspect of the OOA Method

The essential features recorded in Table 4.1 reflect the 'what' aspect of the OOA method. They show that the aim of this method is to build a five-layer OOA model of a system.

Action 2.1.1: Assess the Relationship between the Features in the 'What' Aspect

According to the criteria included in this action, the dependencies of the essential features in this part of the OOA method are determined by their meanings and roles, as shown in Table 5.1. The features recorded in the same row (expressed by dotted lines) means that they depend on one another. The implicit features recorded in the table are identified as follows.

— Assess the relationship between 'fundamental concept' and 'model'

By assessing the relationship between 'fundamental concept' and 'model', it is found that the features 'whole-part structure', 'object state' and 'subject' in the column 'model' seem not to really rely on the features in the column 'fundamental concept'. This means, some extra concepts should be implicitly regarded as fundamental by this method, and they could be the concepts 'aggregation', 'object lifecycle' and 'partitioning' (see section 3.6).

— Assess the relationship between 'fundamental principle' and 'fundamental concept'

The fundamental principles behind the above implicit concepts are 'pervading methods of organisation', 'categories of behaviour' and 'scale' recorded in Table 4.1.

— Classify the principle 'abstraction'

The principle 'abstraction' also implies the behaviour abstraction that relates to the behaviour aspect of a system, as the concept 'object lifecycle' is supported in the method.

Principle / Part / Stage	Fundamental Principle	Fundamental Concept	Model	
			Element	Type
Analysis	•Abstraction (data, behaviour)	•Object •Class	•Class-&-Objects •Class	•OOA model
	•Encapsulation (information hiding) •Pervading methods of organisation •Categories of behaviour	•Association •(Aggregation) •(Object lifecycle)	•Attribute •Service •Instance connection •Whole-part structure •Object state	
	•Inheritance	•Inheritance relationship	•Gen-Spec structure (single, multiple)	
	•Communication with messages	•Message	•Message connection	
	•Scale	•(Partitioning)	•Subject	

Table 5.1 The Dependency of the Essential Features in the 'What' Aspect of the OOA Method

Action 2.1.2: Assess the Content of the 'What' Aspect

By considering the meanings and relationships of the essential features that reflect the 'what' aspect of the OOA method, the content of this aspect can be clarified as follows.

• Fundamental Principles

— Principles on object orientation

In the OOA method, the general idea of analysis is to encapsulate data and processes into objects and to share similarities of objects. The principles 'abstraction' (data and behaviour), 'encapsulation', 'pervading methods of organisation' and 'communication

with message' are in particular regarded as fundamental by the method in order to identify and specify objects and classes, their structures and their interactions. The principle 'inheritance' enables the analyst to abstract the similarities of objects. The principle 'scale' enables the analyst to cope with the complexity of a system.

— *Abstraction on data and behaviour*

In view of the row containing 'abstraction' and so on, the data and behaviour aspects of systems are emphasised by this method.

• Fundamental Concepts

— *Concepts around objects*

All fundamental concepts in the OOA method are defined around objects.

— *Concern with object relationship*

The concept 'association' is addressed by this method, focusing on the description of the relationships between objects.

— *Message as basic to object interaction*

The concept 'message' is important for the method in order to express the interactions between objects.

• Object Model

A single object model is defined by the OOA method to specify an object-oriented system. This model consists of five layers each of which describes one aspect of the system.

— *Emphasis on data abstraction*

The model strongly emphasises the abstraction of the data aspect of systems. In particular, the element 'attribute' is provided to specify the data aspect of a system

explicitly. The element 'service' represents the actions performed on 'attribute' and the element 'object state' in the model represents the changes of attribute values over time.

— *Emphasis on data structure rather than process structure*

The concept 'object' is built upon the data abstraction of the OOA method. The concept 'association' is also used by the method to represent the relationships between objects. The specific element 'instance connection' is provided to represent such a structure.

— *A specific structure 'subject layer'*

In order to realise the principle 'scale', the OOA model includes the element 'subject' to give an overview of a part of an object-oriented system, so that a large and complex system is readable.

— *No event or action specified for object state*

The OOA model only includes the element 'object state' to describe the states and state transitions in an object, since the model only focuses on the classification 'change over time' but not 'event-response' in the concept 'categories of behaviour'.

— *Algorithm specified for each service*

The algorithm for each service in an object is required in an OOA model. This may be considered as violating the distinction between analysis and design considered in Chapter 1: analysis is only concerned with 'what' a system is to do but not 'how' to do it.

5.1.2 Activity 2.2: Assess the 'How 'Aspect of the OOA Method

The essential features that reflect the 'how' aspect of the OOA method were identified in activity 1.2, as recorded in Table 4.2. These features are assessed below in detail.

Action 2.2.1: Assess the Relationship between the Features in the 'How' Aspect

To understand the 'how' aspect of the OOA method, the relationship between the features should be considered and assessed, using the criteria given in this action. Figure 5.1 shows an overview of the relationships between the features in the 'how' aspect of the OOA method, taken from Table 4.2.

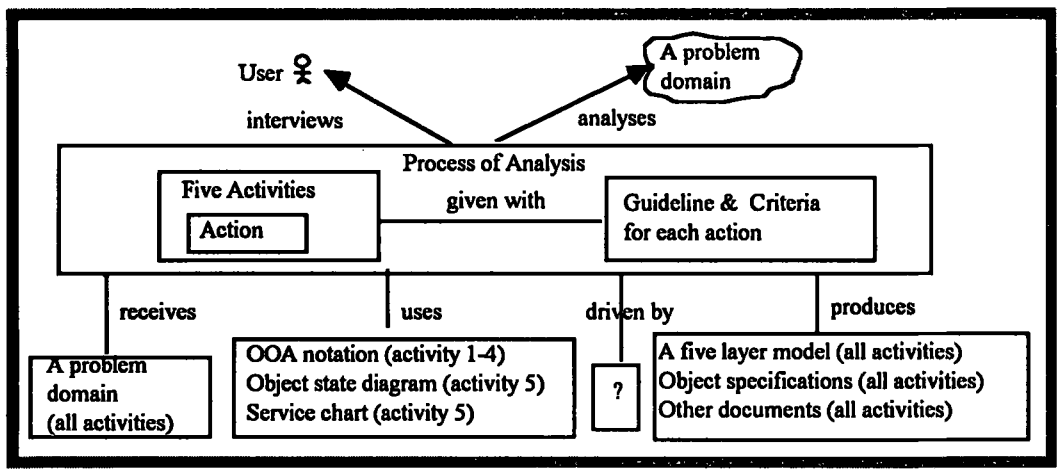

Figure 5.1 The Relationships between the Features in the 'How' Aspect of the OOA method

Action 2.2.2: Assess the Content of the 'How' Aspect

The content of the 'how' aspect of the OOA method is clarified as follows by considering the roles of the features and their relationships.

• Notation

A specific OOA notation is provided by the method to represent the OOA model. In addition, object state diagrams and service charts are also used to specify the services that act on the attributes of objects and classes.

• Process of Analysis

— *Activities rather than steps*

The process consists of five activities rather than steps since no priority is assumed on the construction of five layers in the OOA model.

— *Detailed guidelines and criteria*

The OOA method provides detailed guidelines and criteria for assistance with the process of analysis.

— *Encourage analysts to communicate with users*

This process encourages analysts to contact the users during analysis.

— *Emphasis on data specification*

The process of analysis in the OOA method defines the services in terms of the attributes of objects and classes.

5.1.3 Activity 2.3: Assess the Relationship between Two Aspects of the OOA method

This assessment focuses on the relationships between 'model' in the 'what' aspect and the features in the 'how' aspect of the OOA method.

Action 2.3.1: Assess the Relationship between 'Model' and 'Notation'

In order to represent the five-layer model, the notation provided by the OOA method looks like an extension of entity-relationship diagrams. Other notations, object state diagrams and service charts, are also used. Specific symbols are included in this notation to specify the elements, such as 'class-&-objects', 'subject' and 'service', as shown in Figure 4.1-4.3.

Action 2.3.2: Assess the Relationship between 'Model' and 'Tactic of Analysis'

The OOA model emphasises the static structure of objects within a system. Therefore a *data-driven tactic of analysis*, as an implicit feature, is appropriate for this method, since the process of analysis identifies and specifies the attributes of class-&-

objects first and then the services are defined to manipulate those attributes. This tactic of analysis is added in Table 4.2, and the new version of the table is given in Table 5.2.

Action 2.3.3: Assess the Relationship between 'Model' and 'Process of Analysis'

— *Five layers with five activities*

An OOA model consists of five layers, and the process of analysis in this method includes five activities that build these layers. Activity 1, however, should be carried out first in analysis since the class-&-object layer is the basis of other layers. Actions and associated guidelines and criteria are provided to specify the elements in these five layers.

— *Limitation of the element 'class-&-objects'*

According to the guidelines and criteria given by this method, a 'concept' in a problem domain is not abstracted as a class-&-objects. This may miss some meaningful class-&-objects for an OOA model or cause difficulties in defining useful and intuitive class-&-objects. For example, 'order' in the book trader scenario may be regarded by different analysts as either a physical thing or a concept. In the latter case, it is difficult to abstract 'order' as a class-&-objects from the scenario, if the guidelines are followed strictly.

Action 2.3.4: Assess the Relationship between 'Model' and 'Product of Analysis'

The OOA model is the major product that is produced by the OOA method through OOA.

Practice Part / Stage	Notation	Tactic of Analysis	Input	Process of Analysis			Product
				Activity	Action	Guideline & Criterion	
Analysis	•OOA notation •Object state chart •Service chart •Class-&-objects specification template	Analyse a problem domain by •(Data-driven)	•A problem domain in any style (e.g. problem statements, or dialogue)	•Finding class-&-objects	•Where to look •What to look for •What to consider and challenge	•Observe first-hand, listen actively, ... •Looking for structures, other systems,..... •Needed rememberance, needed behaviour,...	•A five-layer model •Class-&-object specifications •Other documents, if needed
				•Identifying structures	•What to look for •What to consider and challenge	•Consider each class as a generalisation,..... •Check if it is in the problem domain	
				•Identifying subjects	•Select subjects •Refine subjects •Construct subject layers	•Promote the uppermost class in each ..... •Use problem sub-domain and consider ..... •For a large model, consider using a	
				•Defining attributes	•Identify attributes •Position attributes •Identify instance connections	•What do I need to know? ..... •Apply inheritance in Gen-Spec structure,... •For each object, add connection lines	
				•Defining services	•Identify object states •Identify required services •identify message connections •Specify the services	•Examine the potential values for ..... •Look for the calculations which ..... •Draw an arrow from this object to ..... •Draw services charts •Use specification template with a	

Table 5.2 The Essential Features in the 'How' Aspect of the OOA Method

5.2 Stage 2: Assess the OMT Method (Rumbaugh et al.)

This assessment stage is based on the essential features of the OMT method that were identified during the first stage in section 4.2.

5.2.1 Activity 2.1: Assess the 'What' Aspect of the OMT method

The essential features that reflect the 'what' aspect of the OMT method were recorded in Table 4.3. They show that this method aims to build three kinds of model during analysis (i.e., object model, dynamic model and functional model).

Action 2.1.1: Assess the Relationship between the Features in the 'What' Aspect

The dependencies of the features in the 'what' aspect of the OMT method are decided by their roles and meanings, according to the criteria for this action.

— Assess the relationship between 'fundamental concept' and 'model'

In Table 4.3, the elements in the functional model represent the functions of a system. However, no concept in the column 'fundamental concept' of Table 4.3 emphasises the functions of system. This means that there is an implicit concept, which we call 'system function', that should be regarded as fundamental for the OMT method.

— Assess the relationship between 'fundamental principle' and 'fundamental concept'

See Table 5.3.

— Classify the principle 'abstraction'

The fundamental principle 'abstraction' recorded in the first column of Table 5.3 in fact implies data, process and behaviour abstraction, although data abstraction is most dominant in the OMT method, with dynamic and functional models being defined for the object model.

Based upon this assessment, the relationships between the features in the 'what' aspect of the OMT method are shown in Table 5.3.

Principle / Part / Stage	Fundamental Principle	Fundamental Concept	Model	
			Element	Type
Analysis	•Abstraction (data)	•Object •Classification	•Object •Class •Abstract class	•Object Model
	•Encapsulation (also information hiding) •Combining data and behaviour •Emphasis on object structure not procedure structure	•Object structure	•Attribute •Operation •Association, link •Aggregation •Constraint •Module	
	•Inheritance	•Inheritance relationship	•Generalisation (single, multiple)	
	•Abstraction (behaviour)	•Object behaviour	•Event •Object state •Activity •Action	•Dynamic Model
	•Abstraction (process)	•(System function)	•Process •Data flow •Actor •Data store	•Functional Model

Table 5.3 The Dependency of the Essential Features in the 'What' Aspect of the OMT Method

Action 2.1.2: Assess the Content of the 'What' Aspect

By considering the meanings and roles of the essential features that reflect the 'what' aspect of the OMT method, the content of this aspect is clarified further as follows.

• Fundamental Principles

— Principles of object orientation

The OMT method regards the principles (i.e., 'abstraction' (data, process and behaviour), 'encapsulation' (also information hiding) and 'combine data and

behaviour') as fundamental. With these principles, the method can define objects and classes, model the data, process and behaviour aspects of an object-oriented system, hide the inside of objects from outside and combine data and behaviour into objects. The principle 'inheritance' is also used for sharing the commonalty of objects and classes.

— *Data abstraction as primary abstraction*

Data abstraction is the primary abstraction in the OMT method and other abstractions are used in connection with this abstraction.

— *Object structure as basic structure of a system*

Object structure is regarded essential to modelling a system with the OMT method, to make the system stable.

• Fundamental Concepts

— *Concepts around objects*

All fundamental concepts in the OMT method are defined around objects.

— *Object structure on object relationships*

The concept 'object structure' is considered by the method to be focused on the relationships between objects and the constraints on them.

— *Concern with association but not aggregation*

The concept 'object structure' emphasises the association rather than the aggregation of objects in this method; in most cases an aggregation is just considered as an association with extra properties.

— *Concern with object behaviour*

The behaviour of objects is regarded as important to OMT; with the particular concept 'object behaviour' being addressed by the method.

— Concern with system function

The functions of a system are also specified by the method.

• Models

— Three kinds of model

The OMT method defines three kinds of model to describe object structure, object behaviour and system function, respectively. The object model is basic for defining objects and classes and their attributes and relationships in a system. The other models are built upon it.

5.2.2 Activity 2.2: Assess the 'How' Aspect of the OMT Method

The essential features listed in Table 4.4 reflect the 'how' aspect of the OMT method. They show that, in order to build the three kinds of model, the method provides a process of analysis composed of five steps, each with a sequence of substeps that include the guidelines and criteria for transforming the problem statements into the products of analysis.

Action 2.2.1: Assess the Relationship between the Features in the 'How' Aspect

The relationships among the essential features for the 'how' aspect of the OMT method can be derived from the meanings and roles of the features of the method. They are illustrated in Figure 5.2.

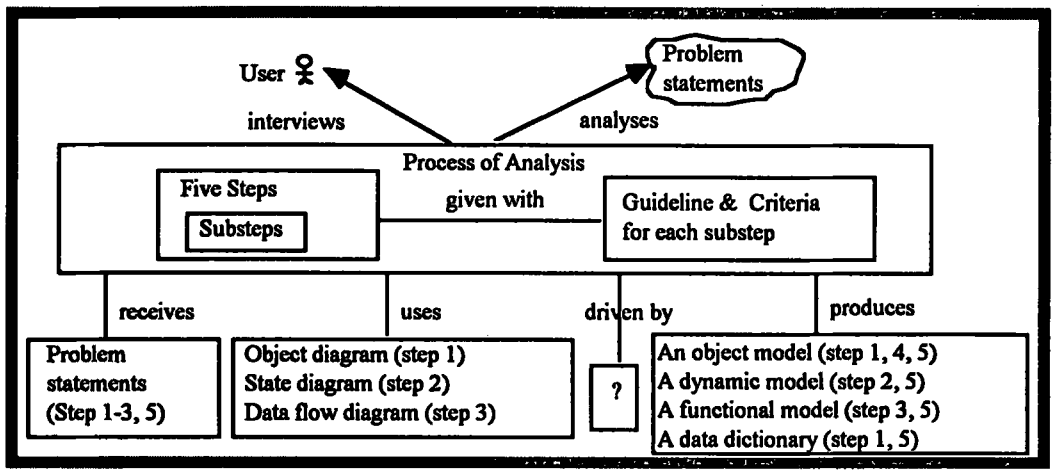

Figure 5.2 Relationships between the Features in the 'How' Aspect of the OMT method

Action 2.2.2: Assess the Content of the 'How' Aspect

The content of the 'how' aspect of the OMT method is clarified as follows by examining the meanings and relationships, according to the criteria given for this action.

• Notation

The major notations that represent the three kinds of models include object diagrams, which are an extension of entity-relationship diagrams, state diagrams and data flow diagrams.

• Process of Analysis

— *Steps and substeps*

The process of analysis consists of a collection of steps and substeps.

— *Detailed criteria and guidelines for identifying attributes*

Very detailed criteria and guidelines are given to define the attributes of objects precisely and correctly. For example, an attribute of an object should be considered as a pure data value, not an object.

— *Weak guidelines and criteria for identifying other elements*

No detailed guidelines and criteria are given to help to determine the meaningful objects and classes, nor to identify the significant operations for objects and classes. In addition, it does not show how to group classes into modules.

— *Iterating analysis and refining the object model*

After an initial version of an object model is constructed, the process encourages iteration to refine the model.

— *Encourage talk with users*

This process requires talking with users during analysis in order to complete the problem statements.

5.2.3 Activity 2.3: Assess the Relationship between Two Aspects of the OMT Method

This assessment focuses on the relationships between 'model' in the 'what' aspect and the features in the 'how' aspect of the OMT method.

Action 2.3.1: Assess the Relationship between 'Model' and 'Notation'

Three types of notation, i.e., object diagram, state diagram and data flow diagram, are used in the OMT method to represent three kinds of model.

Action 2.3.2: Assess the Relationship between 'Model' and 'Tactic of Analysis'

During analysis, the OMT method defines three separate models to specify different aspects of a system. The dynamic model and the functional model are defined in terms of the object model. A data-driven tactic of analysis is therefore undertaken to support the emphasis of this approach. Table 4.4 can be updated by adding this tactic, giving Table 5.4.

Action 2.3.3: Assess the Relationship between 'Model' and 'Process of Analysis'

— *Three steps for constructing the three models*

Three steps are included in the process of analysis for each of the three models in the OMT method. Individual elements in each model are specified in separate substeps in the step.

— *Emphasis on object model*

The object model is regarded as basic to analysis in this method. The first step of this process is thus to build the object model. The dynamic and functional models are then constructed, based upon the object model.

— *Operations upon three kinds of models*

The operations specified in the object model correspond to the queries about attributes or association in the object model, the events in the dynamic model, and the functions in the functional model. Thus, a specific step 'adding operations' is included in the process to specify the operations of classes and objects by the cross-references between these three models.

Action 2.3.4: Assess the Relationship between 'Model' and 'Product of Analysis'

Since the method aims to build three kinds of model for a system, these models represents the products of OMT analysis.

Practice Part / Stage	Notation	Tactic of Analysis	Input	Process of Analysis			Produc
				Step	Substep	Guideline & Criterion	
Analysis	•Object Diagram •State Diagram •Data Flow Diagram	Analyse the problem statements by •(Data-driven)	•A problem statements •dialogue with experts	•Constructing the object model	•Identify objects and classes •Prepare a data dictionary •Identify associations •Identify attributes •Identify inheritance •Test access paths •Iterate and refine the model •Group object classes into modeules	•Consider the nouns in the statements..... •Contain all objects and classes •Condier the verbs and verb phrases in ..... •Consider the nouns followed by ..... •Generalise common aspects of existing ... •Test paths according to the problem ..... •Sign of missing associations, signs of ..... •Group a logical subset of the object model...	•A data dictionary •An object model •A dynamic model •A functional model
				•Constructing the dynamic model	•Prepare a scenario •Identify events between objects •Draw a state diagram •Match events for verification	•First prepare the scenario for "normal case".. •Examine the scenario: all signals, input, ... •If an object receives and sends events ..... •Every event should have a sender and a	
				•Constructing the functional model	•Identify input and output values •Build data flow diagrams •Describe functions •Identify constraints •Specify optimisation criteria •Operations from the object model	•Begin by listing input and output values ... •Working forward or tracing forward ..... •Each function is described in an ..... •Preconditions can be described as the ..... •The values need to be maximised, ... •The operations for reading and writing	
				•Adding operations	•Operations from events, actions and activities •Operations from functions •Conder shopping list operations •Simplify operations	•Events should not be explicitly listed in the object model and they are best represented ... •Put the processes as operations in the object.. •Consider the operations that may be needed .. •Use inheritance where possible to reduce ...	
				•Iterating the analysis	•Refine the analysis model •Restate the requirements	•To refine object definitions to increase ... •Check and confirm the requirements and ...	

Table 5.4 The Essential Features in the 'How' Aspect of the OMT Method

5.3 Stage 2: Assess the Booch Method

This assessment stage is based on the essential features of the Booch method that were identified during the first stage in section 4.3.

5.3.1 Activity 2.1: Assess the 'What' Aspect of the Booch Method

The assessment of the 'what' aspect of the Booch method focuses on the assessment of the features recorded in Table 4.5, and their relationships.

Action 2.1.1: Assess the Relationship between the Features in the 'What' Aspect

The dependencies of the essential features in the 'what' aspect of the Booch method are considered here, according to the criteria given in section 3.5.2.1.

— Assess the relationship between 'fundamental concept' and 'model'

Elements in models and fundamental concepts in the Booch method are classified according to their correspondence, as shown in different rows of Table 5.5.

— Assess the relationship between 'fundamental principle' and 'fundamental concept'

From these relationships we can see that the concept 'object structure' also involves objects collaborating by sending messages to one another. Therefore, another principle, namely 'communicate with messages', should be an implicit principle of the Booch method. This principle is added in the same row with 'object structure' of Table 5.5.

— Classify the principle 'abstraction'

In the Booch method the principle 'abstraction' is used to emphasise the operations of objects and their classes, based on the processing and behavioural aspects of systems. This implies that both process abstraction and behaviour abstraction are important in the Booch method, as shown in Table 5.5.

Principle / Part / Stage	Fundamental Principle	Fundamental Concept	Model	
			Element	Type
Analysis	•Abstraction (process, behaviour)	•Object •Class	•Object •Class •Abstract class	•Object Model
	•Encapsulation (also information hiding) •(Communication with messages) •Hierarchy (with inheritance & structuring) •System decomposition	•Object structure •Class structure •Subsystem	•Object state •Operation (also message) •Object relationship •Field •Using relationship •Inheritance relationship (single, Multiple) •Module	

Table 5.5 Relationship between The Features in the 'What' Aspect of the Booch Method

Action 2.1.2: Assess the Content of the 'What' Aspect

The content of this aspect is assessed by considering the meanings and roles of the essential features of the 'what' aspect of the Booch method, together with the above relationships.

• Fundamental Principles

— *Principles of object orientation*

The Booch method uses the principle 'abstraction' for describing objects and classes, the principle 'encapsulation' (also information hiding) for determining the visibility of objects, the principle 'hierarchy' for sharing the similarities of objects and classes, the principle 'communication with message' for interacting between objects, and the principle 'system decomposition' for modelling a logical structure of a system.

— *Emphasis on process and behaviour*

The process and behaviour aspects of a system are derived by the principle 'abstraction' (process and behaviour).

• Fundamental Concepts

— *Concepts around objects*

All fundamental concepts in the Booch method are based on objects.

— *Considering objects and classes in different structures*

The concepts 'object structure' and 'class structure' are considered separately in the Booch method.

— *Object structure on object collaboration*

The concept 'object structure' in this method focuses on the collaborations rather than the relationships between objects.

— *Emphasis on behaviour sharing*

The concepts 'object structure' and 'class structure' focus on the connection of objects or classes through sharing of behaviour.

• Object Model

This method builds one object model for a system.

— *Separate description of objects and classes*

In an object model, objects and classes are described separately by different elements, using the concepts 'object structure' and 'class structure' respectively. Objects are connected by messages; classes are connected by relationships.

— *Emphasis on operations*

The sharing of behaviour of objects is based on the operations of an object and class. The other elements in a model are defined as part of these operations; the data aspect of a system is not emphasised in a model.

— *No element to represent associations between objects*

No specific element is included in the model for specifying associations between objects. Instead, the element 'using relationship' is provided to describe collaborations between objects.

5.3.2 Activity 2.2: Assess the 'How 'Aspect of the Booch Method

The 'how' aspect of the Booch method provides a process of analysis to build an object model of a system. The object model is represented by object diagrams, class diagrams and state transition diagrams.

Action 2.2.1: Assess the Relationship between the Features in the 'How' Aspect

The relationships among the essential features in the 'how' aspect of the Booch method are illustrated in Figure 5.3, determined using the criteria for this assessment.

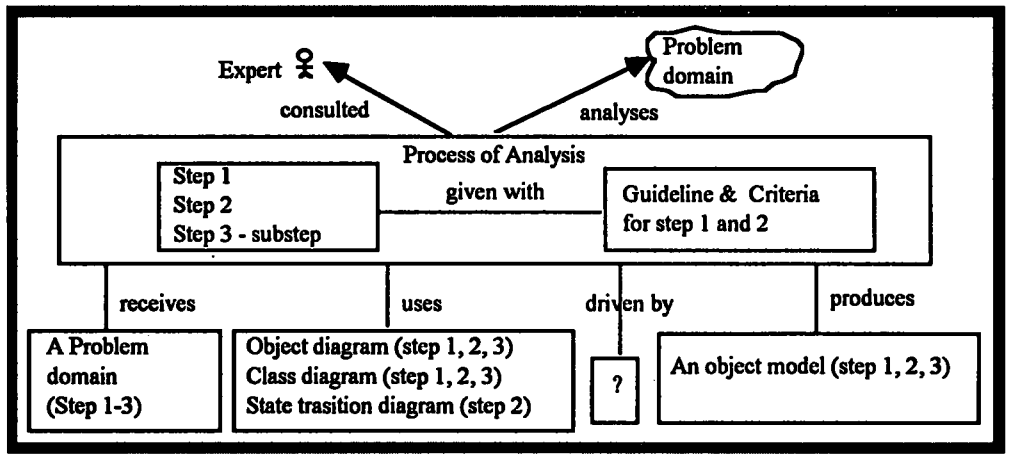

Figure 5.3 The Relationships between the Features in the 'How' Aspect of the Booch method

Action 2.2.2: Assess the Content of the 'How' Aspect

In addition to the description of 'process of analysis' in section 4.3.2, the extra interpretation of these features of the Booch method is as follows, in terms of the given criteria of assessment.

• Process of analysis

— *Analyse a little, design a little*

The fourth step of the process of analysis overlaps with the process of design. It enables prototyping of objects and classes specified in the previous steps.

— *No substeps provided*

This process of analysis includes three analysis steps; however no substeps are included for these steps.

— *No detailed guidelines and criteria given*

The feature 'process of analysis' described in Section 4.3.2 shows that the Booch method does not give any detailed guidelines and criteria to assist with the steps of the analysis stage. The resultant model may be subjective in this situation.

— *Domain experts as consultants*

This process regards domain experts as consultants during analysis, in order to identify the objects that are general and important in a problem domain.

5.3.3 Activity 2.3: Assess the Relationship between Two Aspects of The Booch Method

This assessment focuses on the relationships between 'model' in the 'what' aspect and the features in the 'how' aspect of the Booch method.

Action 2.3.1: Assess the Relationship between 'Model' and 'Notation'

An object model specifies the object structure and the class structure separately, using notations for object diagrams, class diagrams and state transition diagrams.

Action 2.3.2: Assess the Relationship between 'Model' and 'Tactic of Analysis'

An object model in the Booch method describes problem-domain objects that play roles in the application, and the interaction between these objects, by focusing on the functions performed by or on the objects. This means that the tactic of analysis in the Booch method is process-driven. Table 5.6 contain this tactic of analysis so that the features in the 'how' aspect of the method is completed now.

Action 2.3.3: Assess the Relationship between 'Model' and 'Process of Analysis'

— *Emphasis on object structure and class structure*

An object model consists of an object (structure) diagram and a class (structure) diagram for a system. The process of analysis gives three steps to identify objects and classes, to specify their semantics in the system, and to decide the relationships between classes and collaborations between objects.

— *Emphasis on process abstraction*

Since most elements in the object model are related to the specification of operations for objects and classes, the steps of analysis focus on the process abstraction. In particular, the semantics of and the relationships between objects and classes are concerned with the operations and behaviour of the objects and classes.

Action 2.3.4: Assess the Relationship between 'Model' and 'Product of Analysis'

The method builds one object model for a system, and it is thus the product of analysis.

Practice Part / Stage	Notation	Tactic of Analysis	Input	Process of Analysis			Product
				Step	Substep	Guideline & Criterion	
Analysis	•Class Diagram •Object Diagram •State Transition Diagram	Analyse a problem domain by •(Process-driven)	•A problem domain	•Identifying classes and objects		•Analyse the vocabulary of the problem domain by following one of three classification approches	•An object model
				•Identifying the semantics of classes and objects		•Writing a script for each object to define its life cycle, including its characteristic behaviour	
				•Identifying the relationships among classes and objects	•Discover the relationships between the classes and objects •Decide the visibilities between classes and objects		

Table 5.6 The Essential Features in the 'How' Aspect of the Booch Method

5.4 Stage 2: Assess the Wirfs-Brock Method (Wirfs-Brock et al.)

This assessment stage is based on the essential features of the Wirfs-Brock method that were identified during the first stage in section 4.4.

5.4.1 Activity 2.1: Assess the 'What' Aspect of the Wirfs-Brock Method

The essential features recorded in Table 4.7 show that the Wirfs-Brock method aims to specify an object-oriented system by building an object model.

Action 2.1.1: Assess the Relationship between the Features in the 'What' Aspect

The dependencies of the essential features in the 'what' aspect are determined by their roles and meanings in this method, using the given criteria.

— Assess the relationship between 'fundamental concept' and 'model'

In terms of the criteria of assessment, the relationship between these two kinds of features are determined as shown in Table 5.7.

— Assess the relationship between 'fundamental principle' and 'fundamental concept'

The feature 'subsystem' in the column 'fundamental concept' does not depend on any feature in the column 'fundamental principle', according to its meaning and role in this method. An implicit principle should therefore be introduced for this method. We name it 'scale', that means to partition or group logically related things.

— Classify the principle 'abstraction'

The operations (i.e., responsibilities) of objects and classes are particularly emphasised by the fundamental concepts and the object model, according to the meanings of the features described in the first stage of assessment. The principle 'abstraction' in the

Wirfs-Brock method should reefer to the process abstraction on which the above concepts and model can rely.

The dependency of these features on one another in the 'what' aspect of the Wirfs-Brock method is shown in Table 5.7.

Principle / Part / Stage	Fundamental Principle	Fundamental Concept	Model	
			Element	Type
Analysis	•Abstraction (Process)	•Object •Class	•Class •Abstract class	(•Object model)
	•Encapsulation •Information hiding	•Object responsibility	•Responsibility	
	•Inheritance	•Inheritance relationship	•Inheritance hierarchy	
	•Message-sending	•Client-server	•Client-server contract	
	•(Scale)	•Subsystem	•Collaboration •Subsystem	

Table 5.7 The Relationship between the Features in the 'What' Aspect of the Wirfs-Brock Method

Action 2.1.2: Assess the Content of the 'What' Aspect

By considering the meanings and relationships of the essential features that reflect the 'what' aspect of the Wirfs-Brock method, the content of this aspect can be clarified as follows.

• Fundamental Principles

— Principles on object orientation

The Wirfs-Brock method uses the principle 'abstraction' for finding objects and classes; the principles 'encapsulation' and 'information hiding' for integrating data and process into objects and hiding the internal detail of objects from outside; the principle 'inheritance' for sharing the similarities of classes; the principle 'message-sending' for

collaborating objects; and the principle 'scale' for grouping the logically related classes together.

— *Emphasis on process*

The process aspect of an object-oriented system is focused by analysis, according to the principle 'abstraction' (process).

— *Separation of encapsulation and information hiding*

The Wirfs-Brock method regards the principles 'encapsulation' and 'information hiding' as two different principles.

• Fundamental Concepts

— *Concepts around objects*

All fundamental concepts in the Wirfs-Brock method are defined in terms of objects.

— *Not concerned with object relationship*

No concept that emphasises the relationships between objects is considered by the Wirfs-Brock method.

— *Emphasis on the client-servers*

The concept 'client-server' is regarded as fundamental in the Wirfs-Brock method, in order to emphasise the collaborations between objects.

— *Object responsibility*

The concept 'object responsibility' is a particular feature of the Wirfs-Brock method. This concept supports the principle 'abstraction' (process).

• Object Model

This method builds one object model for a system.

— *Emphasis on responsibilities of objects*

This method focuses on object responsibilities. An object model thus includes a specific element 'responsibility' to specify the responsibilities of objects.

— *Not explicitly specify objects*

A model does not explicitly consider objects. Objects are implied as instances of the classes that they belong to.

— *Not define associations between objects*

The associations between objects are not specified for a model.

— *Not specify object states*

Object states are not specified for a model. The responsibilities of classes over time are not specified.

5.4.2 Activity 2.2: Assess the 'How' Aspect of the Wirfs-Brock Method

The essential features that reflect the 'how' aspect of the Wirfs-Brock method were recorded in Table 4.8. These features are assessed below in detail.

Action 2.2.1: Assess the Relationship between the Features in the 'How' Aspect

The relationships among the essential features in the 'how' aspect of the Wirfs-Brock method are based upon the interpretation of the features in section 4.4. An overview of the relationship is shown in Figure 5.4.

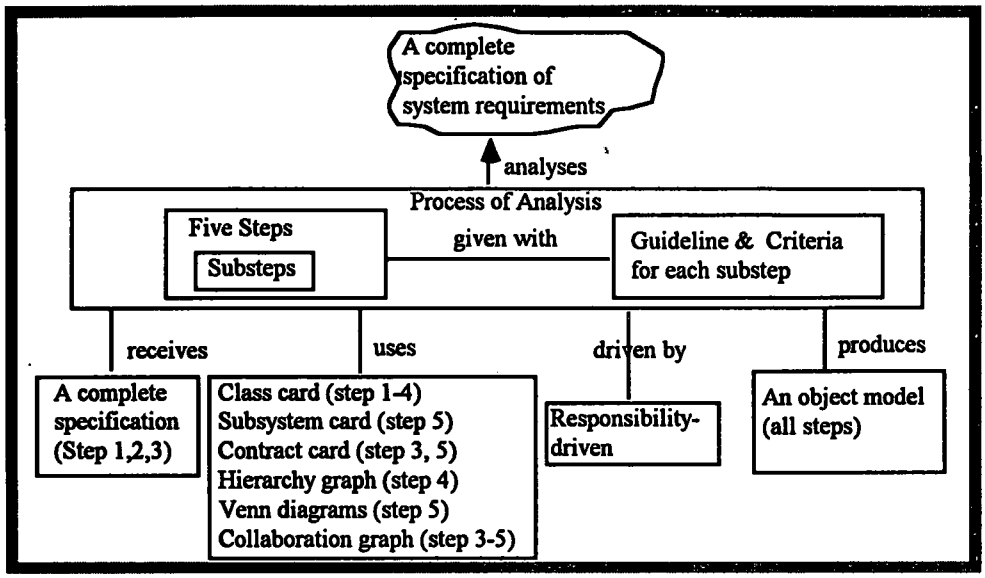

Figure 5.4 The Relationships among the Essential Features in the 'How' Aspect of the Wirfs-Brock method

Action 2.2.2: Assess the Content of the 'How' Aspect

The content of the 'how' aspect of the Wirfs-Brock method is considered again, based upon these relationships between the features.

• Input of Analysis

The Wirfs-Brock method does not encourage analysts to interview users or experts during analysis, as a complete specification of the system requirements has to have been provided before using the method.

• Process of Analysis

— *Emphasis on collaborations among objects and classes*

There are five steps for the process of analysis. However, three steps are closely related to the identification and definition of collaborations between classes.

— *Detailed guidelines and criteria for defining responsibilities*

With the responsibilities of objects being emphasised by this method, detailed guidelines and criteria are given to define these responsibilities.

5.4.3 Activity 2.3: Assess the Relationship between Two Aspects of the Wirfs-Brock Method

This assessment focuses on the relationships between 'model' in the 'what' aspect and the features in the 'how' aspect of the Wirfs-Brock method.

Action 2.3.1: Assess the Relationship between 'Model' and 'Notation'

To represent the elements 'class', 'inheritance hierarchy', 'collaboration' and 'subsystem' in the object model, the notation given in this method includes class card, contract card, venn diagrams, hierarchy graphs, collaboration graphs, and subsystem card respectively.

Action 2.3.2: Assess the Relationship between 'Model' and 'Tactic of Analysis'

An object model focuses on the responsibilities of objects, and so the tactic of analysis in this method is responsibility-driven.

Action 2.3.3: Assess the Relationship between 'Model' and 'Process of Analysis'

— One object model required

The steps of the process of analysis define the elements of a single object model.

— Emphasis on process abstraction

Since an object model focuses on the responsibilities of objects, the process of analysis in the method emphasises the identification and specification of these responsibilities. The collaborations between classes or subsystems in a model are specified in terms of these responsibilities.

Action 2.3.4: Assess the Relationship between 'Model' and 'Product of Analysis'

An object model should be the product of analysis generated by the Wirfs-Brock method.

5.5 Stage 2: Assess the Syntropy Method (Cook and Daniels)

This assessment stage is based on the essential features of the Syntropy method that were identified during the first stage in section 4.5.

5.5.1 Activity 2.1: Assess the 'What' Aspect of the Syntropy Method

The features recorded in Table 4.9 show that the Syntropy method aims to build an essential model and specification model. The first model describes what is involved in a situation in the real world and the second model specifies what the software will do, according to the essential model.

Action 2.1.1: Assess the Relationship between the Features in the 'What' Aspect

— Assess the relationship between 'fundamental concept' and 'model'

Table 5.8 shows the relationship between the concepts and the models, using different rows to represent the dependency.

— Assess the relationship between 'fundamental principle' and 'fundamental concept'

Table 5.8 shows the dependencies of the concepts on the principles, according to the meanings of the principles and concepts given in Table 4.9.

— Classify the principle 'abstraction'

The Syntropy method regards the principle 'abstraction' as data abstraction and behaviour abstraction, so that the data and behaviour aspects of software can be identified

and specified, from which the concepts 'object', 'object type', 'object structure', 'object behaviour', etc. are defined. The implication of the principle 'abstraction' in the Syntropy method is shown in Table 5.8.

Principle / Part / Stage	Fundamental Principle	Fundamental Concept	Model	
			Element	Type
Analysis	•Abstraction (data)	•Object •Object tpye	–Type view •Object •Object type •Abstract type	•Essential Model •Specification Model
	•Encapsulation (also information hiding) •Domain	•Identity •Value •Object structure	•Property •Value type •Association •Aggregation •Constraint •Invariant •Navigation expression •Domain •Domain dependency	
	•Inheritance	•Sub-typing	•Type extension (single, multiple)	
	•Abstraction (behaviour)	•Object behaviour •Event broadcast	–State view •Object state •State type •State invariant •Event •Event scenario •Creation operation •Generation	

Table 5.8 The Relationships between Features in the 'What' Aspect of the Syntropy Method

Action 2.1.2: Assess the Content of The 'What' Aspect

By considering the meanings and relationships of the essential features that reflect the 'what' aspect of the Syntropy method, the content of this aspect can be clarified as follows.

• Fundamental Principles

— *Principles on object orientation*

In order to model a problem situation with object orientation in both type and state views, the principle 'abstraction' with data and behaviour is adopted, focusing on the information about objects, values and events involved in the situation. Additionally, the principles 'encapsulation', 'domain', 'inheritance', and 'event broadcast' are also regarded as fundamental to object modelling.

— *Abstraction with data and behaviour*

The principle 'abstraction' is used by emphasising the data and behaviour aspects of the objects which describe a problem situation.

• Fundamental Concepts

— *Concepts around objects*

All fundamental concepts addressed by the Syntropy method are related to and defined upon objects.

— *Concern with object structure and object behaviour*

The concepts 'object structure' and 'object behaviour' are addressed by this method as they are regarded as the notions to capture objects, values and events, of which any problem situation in the world consists, according to this method.

— *Value as basic notion to show the states of objects*

The concept 'value' is addressed by this method to relate to the state of objects in the real world.

— *Concern with event broadcast*

The concept 'event broadcast' emphasises the communication between objects with events.

• Object Model

In the Syntropy method, two models are built in terms of objects: an essential model and a specification model. An essential model describes a situation in the world with both type and state views; a specification model describes what a software system should do.

— *Essential model and specification model*

These two models describe the data and behaviour aspects of a problem situation and a software system, respectively. A specification model is the refinement of the essential model for the same situation. The elements 'properties', 'invariants', 'constraints', 'event', 'states' (i.e., values of the properties), 'creation operation' and 'generation' are used to specify the details of these two aspects of objects.

— *Emphasis on both data structure and event scenario*

The two models focus on the object structure and the object behaviour. Object structure describes objects and object types, their properties with constraints or invariants, and their relationships, in terms of the elements 'object', 'object type', 'property', 'value type', 'association', 'type extension', etc. An event scenario describes the interactions between objects with event triggering. It also shows how objects change their state and how they respond to different events.

— *Description of invariants*

The element 'invariant' is used to describe constraints on the range of values for both kinds of model.

5.5.2 Activity 2.2: Assess the 'How' Aspect of the Syntropy Method

The essential features that reflect the 'how' aspect of the Syntropy method are recorded in Table 4.10. It shows that the process of analysis supported by this method could be either

data-driven or event-driven. In addition, this method uses the OMT graphic notation, statecharts and formal notation to represent an essential model and specification model.

Action 2.2.1: Assess the Relationship between the Features in the 'How' Aspect

The assessment of the relationships between the features in the 'how' aspect is mainly concerned with the relationship between 'process of analysis' and other features in the aspect. This assessment of the Syntropy method is delayed until after the assessment of 'model' and 'process of analysis' (i.e., in action 2.3.3 below), since the authors of the method do not make this process explicit in their book [Cook94a].

5.5.3 Activity 2.3: Assess the Relationship between Two Aspects of the Syntropy Method

This assessment focuses on the relationships between 'model' in the 'what' aspect and the features in the 'how' aspect of the Syntropy method. In particular, the implicit process of analysis in this method can be decided by this activity.

Action 2.3.1: Assess the Relationship between 'Model' and 'Notation'

An essential and a specification model contain the structure of the objects and the behaviour of the objects for a system. To represent these two aspects of a system, the method adopts OMT notation [Rumbaugh91] and statecharts [Harel87]. The semantics of the models, in particular the properties and their values within objects in the models, is defined by 'invariants' and 'constraints' in a mathematical notation[Hayes87].

Action 2.3.2: Assess the Relationship between 'Model' and 'Tactic of Analysis'

Essential and specification models have both data and behaviour aspects that can be decried individually, so two sorts of tactic of analysis are supported by this method, i.e., data-driven and event-driven; thus the two aspects can be specified concurrently.

Action 2.3.3: Assess the Relationship between 'Model' and 'Process of Analysis'

The process of analysis in the Syntropy method can be inferred according to the content of the models, and is specified below.

— *Two models built by two steps*

Since the Syntropy method needs to build one essential model and one specification model, the process of analysis must includes two steps to construct each of the two models, respectively.

— *Two views of models and two actions*

Both essential models and specification models support two views: a type view and a state view. These two views need to be determined by two actions in each step of the process of analysis: describing object structure in a type view and object behaviour in a state view for an essential model; specifying a system in a type view and in a state view for a specification model.

The further detail of the process of analysis in this method can be described as follows, based upon the relationship between 'model' and 'process of analysis'.

Step 1: Building an Essential Model

Guidelines: Building an essential model aims to establish the facts about a pre-existing situation, or to describe a situation to be constructed. The essential model represents the following details about a situation:

- the possible states for a given situation;
- the set of events which cause changes between one state and another; and
- the possible sequences of events which can occur.

The states of a situation are described in terms of objects, which have properties, and their relationships. In using the Syntropy method, any particular state consists of a set of objects, each with specific properties, participating in particular relationships. Two major actions are included in this step, as follows; either of them can be carried out first:

Action 1: Describing Object Structure in a Type View

In this action, the following elements of the essential model are used to describe the type view.

— *Finding objects and object types (a textual analysis)*

Guidelines: Candidate object types are identified from the problem statement by a textual analysis, i.e., by considering the nouns and noun phrases in the textual description.

Criteria: A 'good' object should not be redundant, an attribute of another object, an operation, or implementation construct. This view should cover all those things considered to be separate and interchangeable. Operations such as complex parameterised algorithms should be modelled as object types.

— *Identifying properties and value types*

Guidelines: The properties of an object type need to be identified by considering the information which each object type knows about. A property of an object can be observed by another object that relates to this object.

— *Identifying associations between objects*

Guidelines: This substep identifies associations between objects. An association should not be a connection of anything with access paths or implementation visibilities.

— *Identifying type extensions*

This substep defines the sub-types by extending the capabilities of the super-type, according to the description of the given situation.

— *Defining constraints and invariants*

Guidelines: In order to model a real-world situation precisely, and allow a common understanding by a group of analysts and designers, the formal precision of mathematics, in the form of set theory and logic, is used in conjunction with a model to specify logical type constraints on associations and invariants on object properties.

— *Defining state types*

State types are identified in this substep, to provide a direct link between the type view and the state view of a model.

Guidelines: (a) A state type must always be a sub-type of a normal type because it represents one possible state for objects confirming to the normal type. (b) Objects cannot be created to conform to a single state type; conformance with state types will change as the object changes state. (c) Not all the possible states need to be included in the type view, but each state type in the type view should correspond to exactly one state on a statechart in the state view.

Criteria: A state type can have sub-types in a nested state structure if needed; if an object is in a state represented by a state type that has sub-types, it must also be in a state represented by one of the sub-types.

Action 2: Modelling object behaviour in State View

Cook and Daniels state in[Cook94a] that the purpose of the essential model is to describe the possible states of the system, the possible sequences of events and how states change when events occur. This substep therefore describes all the ways in which a

situation can change, by defining all the possible sequences of events. That is, if a sequence of events can be observed for a situation, the essential model must indicate that the sequence is valid.

— *Discovering events*

Guidelines: Events can be identified by: (a) systematically considering the object types and their associations; and (b) producing event scenarios, each of which is a sequence of specific event instances showing just one of many possible sequences of events that could occur in a situation. In the first way, for each type, consider how an instance of the type is created, and how it is destroyed. For each association, consider (1) how an instance of it is created; (2) how an instance of it is destroyed; and (3) in the case of an ordered association, how its order is established or changed. An event table which consists of its name, parameters, pre-conditions and sequence can be used to bring together definitions of events.

— *Drawing statecharts*.

The state view of each object type is described by a separate statechart which supports nested states and orthogonal states. The state view of a situation is the combination of the separate statecharts.

Guidelines: Statecharts capture the information: (a) events of interest to an object type; (b) a finite state machine with states, state transitions and state invariants; (c) details of object creation; (d) constraints on the validity of events; and (e) descriptions of event consequences. Each state type in the type view must correspond directly to a state in a statechart. State invariants are specified inside states in a statechart. They are logical expressions which are always true when the object is in a particular state. A statechart

may include two or more concurrent state machines which describe the concurrent states of an object type.

Step 2: Building a Specification Model

An essential model is built in the case where the software boundary is not well-understood, in order to provide a systematic way of making decisions about that boundary. The specification model is then built by extending the essential model, using the same notation, by considering what the software will do. Three actions are included in this step.

Action 1: Drawing a System Boundary

In the specification model, the boundary of a system is decided by specifying incoming events in terms of the changes of state which they cause, and outgoing events generated as a result. A specification model is thus a stimulus-response model.

Guidelines: The first way to determine the boundary is to decide whether each event in the essential model is detected or generated by the software, or is irrelevant to the operation of the software. The second way is to consider untimely occurrences such as exception occurrences and then define detected events representing these occurrences in the specification model. The third way is to consider the interface between the software and its environment in terms of agents that are outside the software itself, who may be people or other systems which interact with the software.

Action 2: Specifying the System in a Type View

Guidelines: The specification model of the system needs be specified in this action, which describes object types in the software rather than in the situation. In order to distinguish

between object types in the essential model and specification model, the symbol '-S' is appended to the type-names in the specification model.

Action 3: Specifying the System in a State View

Every object type in the specification model has a state view which defines how instances of the type respond to events. The syntax for events in this model is the same as that in the essential model. The events in the specification model are instantaneous and broadcast, as in the essential model. In addition, the specification model specifies 'generated events', as follows.

— *Specifying generated events*

Guidelines: Generated events are the events which are produced from specific transitions, or from event list entries. Generated events are the actions in Statecharts[Harel97].

— *Specifying entry and exit generations*

Guidelines: Entry and exit generation is triggered on any entry and exit from the state, including transitions which explicitly begin and end with the same state. Allowed events which do not cause a state transition, however, do not trigger entry or exit generation. Event generations are specified in the event list of the statechart, meaning that the generation occurs whenever the event occurs.

— *Specify internal events*

Guidelines: Internal events are the events for a state view of an object type, generated and detected by the software. Their consequences should be established before any further external events occur.

— *Ordering events*

Guidelines: The states and events in a state view are specified in the order: (1) establish all of the post-conditions for an event to exit and enter the target state; (2) trigger exit generations on the source state; (3) trigger generations defined in the event list; (4) trigger generations on the transitions; (5) trigger entry generations on the target state. If an event is allowed and has no transition defined for the current state, it proceeds by defining the post-conditions and then triggering in order any generations defined in the event list. The object is already in its new state when any generation in it occurs. Any exit generation happens after the object is in its target state.

— *Specifying object responsibilities*

The responsibilities for overall system behaviour are allocated to individual objects in this substep.

Guidelines: To specify object responsibilities, an object can be thought of as having the responsibilities: (a) knowing (i.e., remembering or calculating) a value; (b) listening for an event; (c) telling other objects about an event; (d) creating new objects. In the specification model, these responsibilities correspond to properties, event list entries, generations and object creations.

The detail of the 'how' aspect of the method is now shown in Table 5.9 to replace Table 4.10. In addition, the relationship between the features in the 'how' aspect can be determined, as shown in Figure 5.5.

Practice Part / Stage	Notation	Tactic of Analysis	Input	Process of Analysis			Product
				Step	Action	Guideline & Criterion	
Analysis	•OMT notation (object diagram) •Statechart •Mathematics notation	Analyse a problem statement by •Data-driven or •Event-driven	•A problem statement	•Building an essential model	•Describing a type view	•Finding objects and object types ..... •Identifying properties and value types ..... •Identifying associations between objects... •Identifying type extension ..... •Defining constraints and invariants ..... •Defining state types.....	•An essential model •A Specification model
					•Describing a state view	•Identifying events ..... •Drawing statecharts.....	
				•Building a specification model	•Drawing a system boundary	•Decide whether each event ..... •Consider untimely occurences ..... •Think about the interface.....	
					•Specifying a type view	•Describe object types in the software	
					•Specifying a state view	•Specifying generated events ..... •Specifying entry and exit generations ..... •Specify internal events ..... •Odering events ..... •Specifying object responsibilities	

Table 5.9 The Essential Features in the 'How' Aspect of the Syntropy Method

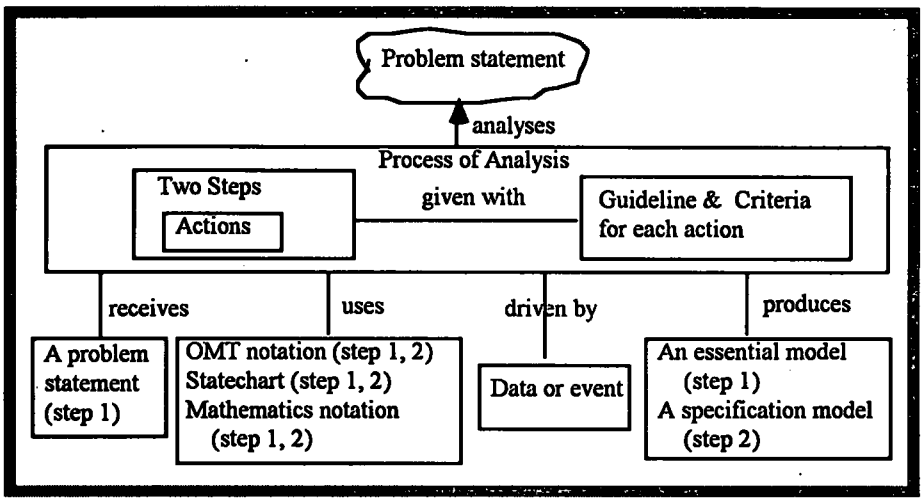

Figure 5.5 The Relationship between the Features in the 'How' Aspect of the Syntropy method

Assess the Content of the 'How' Aspect (i.e., Action 2.2.2)

We can now return to the assessment of the 'how' aspect of the Syntropy method, based on the relationships shown in Figure 5.5

• Process of Analysis

— *Two steps to build two models in sequence*

The process consists of two steps which construct an essential model and a specification model in sequence.

— *Some guidelines and criteria*

The Syntropy method provides some guidelines and criteria to assist with the construction of two models.

— *Not force the analyst to communicate with the user*

This process does not require a dialogue with the user during analysis.

— *Support for two views*

The process of analysis in the Syntropy method supports two views to model a problem situation and specify a software system: a type view and a state view.

Action 2.3.4: Assess the Relationship between 'Model' and 'Product of Analysis'

It can be seen that the product of analysis delivered by the Syntropy method comprises an essential model and a specification model.

5.6 Understanding of the Five Analysis Methods

In the above sections, five analysis methods are assessed in terms of the framework defined in Chapter 3. This section discusses general features of these five methods based on the results of above assessment.

5.6.1 Understanding of the 'What' Aspects of the Methods

(1) Fundamental Principles

— *Abstraction*

The five methods all regard the principle 'abstraction' as fundamental, but they each use it with a different emphasis: data and behaviour abstraction in the OOA method and Syntropy methods; data, process and behaviour abstraction in the OMT method; the process and behaviour abstraction in the Booch method; and the process abstraction in the Wirfs-Brock method.

— *Encapsulation and Information Hiding*

These methods use the principles 'encapsulation' and 'information hiding' in different ways: the methods, with the exception of the Wirfs-Brock method, use them

interchangeably; that is, to hide the internal detail of an object from other objects. The Wirfs-Brock method uses the two terms differently; that is, 'information hiding' has the same meaning as for other methods, whereas 'encapsulation' means to combine data and process into objects.

— *Inheritance*

The principle 'inheritance' is used by these methods for analysing the similarities of classes in a system.

— *System decomposition*

The methods use this principle such as 'scale' in the OOA method and 'domain' in the Syntropy method to partition, or group together, the objects and classes that are logically related to one another.

— *Communication with messages*

The OOA method, the Booch method and the Wirfs-Brock method consider the principle 'communication with message' as fundamental for analysis.

(2)Fundamental Concepts

— *Concepts around objects*

All fundamental concepts in these methods are defined around objects, since the fundamental principles are used to support object orientation.

— *Object and Class*

The concepts 'object' and 'class' (also 'classification' in the OMT method and 'object type' in the Syntropy method) are addressed by these methods: in general, 'object' means a thing, a concept, or an entity in a problem domain or an application scenario; 'class' means a set of objects that have common properties and behaviour.

— *Inheritance relationship*

These methods regard the concept 'inheritance relationship' between classes as fundamental.

— *Object structure*

All the methods except the Wirfs-Brock method regard 'object structure' as fundamental. However, the meaning and the role of this concept in the Booch method are different from the OOA method, the OMT method and the Syntropy method.

— *Subsystem*

'Subsystem' (or 'partitioning' in the OOA method, or as implied in 'object structure' in the OMT method) is regarded as significant for analysis.

— *Object behaviour*

'Object behaviour' is addressed by all the methods except the Wirfs-Brock method.

— *Association*

The concept 'association' (also implied in 'object structure' in the OMT method) is addressed by the OOA method, the OMT method and the Syntropy method. This concept expresses the relationships between objects.

— *Aggregation*

Only the OOA method regards the concept 'aggregation', specifying the whole-part structure of objects, as fundamental.

— *Message*

The concept 'message' (implied in 'object structure' in the Booch method and 'client-server' in the Wirfs-Brock method) is used by the OOA method, the Booch method and the Wirfs-Brock method.

— *Class structure*

'Class structure' is regarded as a fundamental concept in the Booch method, since it considers object structure and class structure separately.

(3)Model

— *Object model*

These methods all aim to build an object model of a system. The model, however, is defined differently in each method since they focus on the different aspects of object-oriented systems.

— *Essential model and specification model*

The Syntropy method builds two different level models: an essential model and a specification model. The former describes the things and concepts in a situation in the world; the latter states what the software will do according to the essential model.

— *Dynamic model and functional model*

The OMT method aims to build another two kinds of model (i.e., a dynamic model and a functional model) of a system; each model is responsible for representing one aspect of the system. The OOA method and the Booch method represent the behaviour aspect of a system by an object model.

The Syntropy method specifies a system in two views: a type view and a state view. In a state view, object behaviour in a system is modelled and specified, which is the same as the dynamic model in the OMT method.

— *Objects and classes*

These methods all define the elements 'object' and 'class' in their model. However, 'object' has different meanings in these methods since the methods focus on and model

different aspects of an object-oriented system. In particular, the OMT method even defines an object without attributes or operations.

— *Abstract classes*

All models in these methods include the element 'abstract class' (or 'class' in OOA and 'abstract type' in Syntropy).

— *Operations of objects and classes*

These methods provide the element 'operation' (or 'service' in OOA, 'responsibility' in the Wirfs-Brock method) in their model to specify the operations of objects and classes explicitly.

— *Attributes of objects and classes*

The attributes of objects and classes are explicitly specified in the object models of the OOA method, the OMT method and the Syntropy method.

— *Object behaviour*

The behaviour of objects is not explicitly specified by the Wirfs-Brock method as it does not support the description of the state of objects and the sequencing of responsibilities (i.e., operations) in objects.

— *Relationships between objects*

The elements 'instance connection' in the OOA method and 'link' in the OMT method and the Syntropy method are provided to specify the relationships between objects.

— *Inheritance relationships between classes (single, multiple)*

These methods include the element 'inheritance relationship' (single and multiple), expressed as 'Gen-Spec structure' in the OOA method, 'generalisation' in the OMT method, 'inheritance hierarchy' in the Wirfs-Brock method, and 'type extension' in the Syntropy method.

— *Communication between objects*

The communications between objects are explicitly specified by the elements 'message connection' in the OOA method, 'event' in the OMT method and the Syntropy method, 'message' in the Booch method and 'client-server contract' in the Wirfs-Brock method.

— *Subsystem*

The element 'subsystem', expressed as 'subject' in the OOA method, 'module' in the OMT method and the Booch method, and 'domain' in the Syntropy method, is used in these methods.

— *Using relationship in the Booch method*

An element 'using relationship' is provided by the Booch method to represent a collaboration between two objects, together with the degree of the collaboration (e.g., one-to-one, one-to-many).

— *Invariant in the Syntropy method*

An element 'invariant' is used by the Syntropy method to represent constraints on data, using a mathematical notion.

5.6.2 Understanding of the 'How' Aspects of the Methods

(1)Notation

Different notations are used by these methods because of the different needs of the models in the methods. Graphical notations are commonly used by these methods. Some notations are the same, or extensions of, conventional notations; this is the case for the notation for object diagrams in the OOA, OMT and Syntropy methods, and the data flow diagram in the OMT method and the state transition diagram in the Booch method. In

addition, the Syntropy method focuses strongly on the constraints on data values and object structure, and uses mathematics notation to represent these constraints.

(2)Tactic of Analysis

Generally, a data-driven tactic of analysis is used by the OOA method, the OMT method and the Syntropy method; the Syntropy method also allows an event-driven tactic. A process-driven tactic is used by the Booch method and the Wirfs-Brock method.

(3) Input of Analysis

For the input of analysis, the OOA method and the Booch method require a problem domain; the OMT method and the Syntropy method assume a problem statement; and the Wirfs-Brock method needs a complete specification of system requirements. In addition, except the Wirfs-Brock method, the methods advise communication with users or domain experts during analysis in order to complete the input of analysis.

(4)Process of Analysis

— *Common roles*

All five object-oriented methods have a generally similar process of analysis, as follows:

— *identify objects and classes*
— *identify operations of objects and classes,*
— *identify inheritance relationships between classes.*

— *Iterative process*

The OMT method and the Booch method recommend an iterative process.

— *Activities or steps of the process*

The OOA method specifies its process as a group of activities that can be carried out in any order; while the other methods emphasise a sequential process with a collection of

steps. Substeps (actions in the OOA method and the Syntropy method) are also provided by the OMT method and the Wirfs-Brock method to reduce the complexity of the analysis.

— *Guidelines and criteria*

Guidelines and criteria are given by these methods (even though there are very few in the Booch method), to assist in building the models. For example, the OOA method has the guideline 'where to look objects and classes' and the criteria 'needed remembrance' of objects. However, the guidelines and criteria given in these methods are different because of the differences between models in the methods. For example, the OOA method basically identified objects and classes by considering the entities in a problem domain, but the OMT method, the Wirfs-Brock method and the Syntropy method basically identified them by looking for the nouns or noun phrases from the problem statements.

— *Check of specifications*

Cross-references between different models is recommended for checking the consistency of three models in the OMT method. Prototyping is required in the Booch method to check an object model. In the Wirfs-Brock method, an object model is checked by a walk-through.

— *Two-level analysis model*

Two-level analysis model is built around objects by the Syntropy method: essential model at top-level and specification model at second level.

(5)Products of Analysis

Different products are produced by the five methods as the models are different.

5.6.3 Conclusions from the above Understanding

The above discussion shows that object-oriented analysis methods carry out analysis in different ways. The conclusions that may be derived from the above understanding of the five methods are as follows:

— *Different 'what' and 'how' aspects*

In general, the contents of 'what' and 'how' aspects of the five methods are different. In particular, although they are all object-oriented they aim to do different things during analysis.

— *Emphasis on analysis or design*

The OOA method, the OMT method and the Syntropy method clearly separate analysis and design, while the other two methods are more focused on object-oriented design and they do analysis and design without an obvious separation.

— *Same terms with different meanings, or vice versa*

The methods use some terms, such as 'object', with different meanings, and use different terms with the same meaning (e.g., 'generalisation' in the OMT method, 'type extension' in the Syntropy method and 'inheritance relationship' in the Booch method).

— *Impacts of the methods on applications*

The 'what' and 'how' aspects of a method have impact on the kinds of application for which the method is used. The results given by the above assessment of these methods enable us to conclude that (1) the OOA method, the OMT method and the Syntropy method are appropriate for modelling information systems since they focus more on data than process in a system; (2) the OMT method, the Syntropy method and the Booch method are appropriate for modelling real-time systems because of their emphasis on the process and behaviour aspects of systems; and (3) the Wirfs-Brock

method is suitable to analyse and model the systems that focus on processes and the collaborations of the processes.

— *First generation and second generation method*

S. Cook and J. Daniels indicate in their book and paper [Cook94a and b] that the Syntropy method is a second generation object-oriented method that was developed by combining the techniques used in first generation object-oriented methods: the OMT method and the Booch method. By assessing this method in terms of our framework, it is seen that this method extends part of OMT techniques (i.e., object modelling and dynamic modelling) giving a type view and state view of a system by adding more elements to the models. It is however not obvious that it also uses the object techniques in the Booch method as claimed in their book [Cook94a].

Chapter 6

Conclusions and Future Work

This chapter includes two sections. The first section initially summarises the work which we have achieved in our research and presented in this thesis; it then identifies the specific contributions of this work to the subject of object-oriented analysis methods. The second section proposes other potential uses of the framework presented in this thesis, as well as indicating possible future research based on this work.

6.1 Conclusions

As discussed in Chapter 1, because of different concerns about specification and construction of computer systems, object-oriented analysis and design must be distinguished by different strategies and heuristics during system development. As we identified in the first chapter, object-oriented analysis is normally concerned with specifying what a system needs to do by focusing on problem domains or users' requirements, whereas object-oriented design emphasises the construction of systems, based on specifications generated by object-oriented analysis. Object-oriented analysis identifies objects from a problem domain or users' requirements, but object-oriented

design develops those objects for a computerised system. An object analysis model (i.e., specification of a system) produced by a method has to be deliverable to design in order to build an object design model (i.e., architecture of the system) [Rumbaugh94]. Because of the differences between object-oriented analysis and design, we must consider and assess a method by separating the aspect of object-oriented analysis from the aspect of design, in order to understand the method precisely. To distinguish between object-oriented analysis and design methods, any method that takes object-oriented analysis into account during system development is referred to as an object-oriented analysis method in this thesis, no matter how it is named in the publication. For example, the Wirfs-Brock method is referred to as an object-oriented analysis method in the thesis, even though it was published as a design method [Wirfs90], since it also covers object-oriented analysis in view of its scope.

Thus the work presented in this thesis focuses on research into object-oriented analysis methods but not design methods. We consider that our study of the general characteristics of object-oriented methods should target analysis, since analysis is independent of implementation and so details of the various programming languages that might be used do not have the impact they do on design. Another reason is that analysis must be carried out prior to design in software lifecycle and therefore the output from the research on object-oriented analysis methods can then be beneficial to the research into design methods.

6.1.1 The Problems of Current Research of Object-Oriented Analysis Methods

Our research found that in the area of object oriented methods, at present there is lack of a framework that can solve problems of understanding individual object-oriented analysis

methods objectively and systematically. As shown in Section 1.3, current research is limited to comparative studies of object-oriented analysis methods. These comparative studies cannot be regarded as such a framework, since they have many limitations relating to the understanding of object-oriented analysis methods, as discussed in Section 1.3. For example, the comparative study by Monarchi and Puhr [Monarchi92] (see Section 1.2) regards the following three features as critical to comparing and evaluating object-oriented methods: representation of systems (i.e., notation), analysis and design process, and mechanism for managing the complexity of systems (i.e., hierarchical construct). Even if these features are truly critical to analysis methods, such a study cannot be regarded as a basis for understanding methods since the study did not cover all the essential features of object-oriented analysis methods which were found by our research. In a summary, all present comparative studies have similar limitations and problems as follows:

(1) The comparative studies do not identify the emphasis and specific roles appropriate to distinguishing between object-oriented analysis and design.

(2) The comparative studies do not support the logical relationships among the features being compared, that is, how a feature such as 'model' has impact on another feature such as 'process'.

(3) The comparative studies do not include consideration of the underlying principles that form the basis of a method.

These limitations and problems mean that the approach of the comparative studies could be inappropriate, superficial and incomplete for understanding a method. Because of the need and importance of understanding methods, we have to overcome the above limitations and problems and establish an approach that can support such understanding.

This is the major motivation and goal of our research; that is, why we were initially interested in the work presented in this thesis and why we established the approach—the framework—defined in Chapter 3. This framework provides a vehicle for understanding an object-oriented analysis method in an objective and systematic way.

Another reason for developing our own framework for object-oriented methods is that, as indicated in the first chapter, we cannot directly reuse the frameworks that were built for understanding traditional methods in the past, such as the Olle's framework [Olle88] (detailed comments on Olle's framework were made in Section 1.4). This is because such a framework does not cover the fundamental issues and concepts of object-oriented analysis such as 'object', 'inheritance relationship' and 'encapsulation'. This means that we have to build a new framework which focuses particularly on the issues and concepts addressed and defined by object-oriented analysis methods.

6.1.2 The Framework: a Solution to the Problems

As a solution to the given problems for object-oriented analysis methods, the framework defined in Chapter 3, provides a comprehensive basis for understanding object-oriented analysis methods in an objective and systematic way. The framework was based upon the study of a number of object-oriented analysis methods that were available at the time of our research; in particular, the use of four representative object-oriented analysis methods [Coad91a, Rumbaugh91, Booch91, Wirfs90] is described in Chapter 2. Our study identified the essential features of these object-oriented analysis methods, as they reflect the fundamental assumptions about object-oriented analysis and the nature of the methods, and they then formed the components of the framework. The 'what' aspect includes the features 'fundamental principle' and 'fundamental concept' and 'model'; and the 'how'

aspect contains the features 'notation', 'tactic of analysis', 'input of analysis', 'process of analysis' and 'product of analysis'. The 'what' aspect of an object-oriented analysis method shows what the method involves, and the 'how' aspect represents how the method performs analysis. The classification of these aspects and the features not only represents the essential constructs of an object-oriented analysis method, but also means that an object-oriented analysis method can be partitioned into smaller parts in order to limit the complexity of understanding it. To support use of our framework for analysing a given method, a multi-level process is provided in Section 3.5, consisting of stages (top level), activities (middle level) and actions (low level). In addition, the criteria for using the framework are given at each level of this process; they are critical to identifying and assessing any feature of a method, or the relationship between two features.

During our research, we have successfully used this framework, following the above process, to assess a large number of object-oriented analysis methods. Chapters 4 and 5 showed the details of assessment of five typical object-oriented analysis methods (i.e., the OOA method [Coad91a], the OMT method [Rumbaugh91], the Booch method [Booch91], the Wirfs-Brock method[Wirfs90], and the Syntropy method [Cook94a]). Additionally, the Appendix shows an outline assessment of another five object-oriented analysis methods (i.e., the Shlaer and Mellor method[Shlaer88], OOSE [Jacobson92], OSA [Embley92], Ptech [Martin92a] and HOOD[Robinson92]). These assessments provide examples, and also demonstrate the process, of using the framework to assess any object-oriented analysis method, for the purpose of understanding the method. The results of the assessments show that, by using our framework, the content and context of the essential features of each method can be identified and clarified clearly and efficiently; a complete picture of the method can then be drawn accurately based upon the relationships

among the features. This demonstrates that our framework is very useful and effective in understanding object-oriented analysis methods. Further, the framework can be also regarded as providing an infrastructure for object-oriented analysis methods, enabling any object-oriented analysis method to be evaluated either for procurement or for further development.

6.1.3 Contributions

The contributions of the work described in this thesis are presented below by considering: (i) general contributions of the work to the area of object-oriented analysis methods, and (ii) specific contributions of the framework to the understanding of analysis methods.

(1) General Contributions of Our Research Work

The contributions of our research work to the area of object-oriented analysis methods are outlined as follows:

Contribution 1.1: *Identification of problems of current research into object-oriented analysis methods*

The *first* contribution of our work is discovery of some problems of current research into object-oriented analysis methods; specifically, the limitations of present comparative studies of the methods. As discussed in the first chapter of this thesis, the major limitations of the studies are (i) no separation of analysis features and design features, (ii) no assumption of logical connections between comparative features, and (iii) no emphasis on the understanding of object-oriented analysis methods. To our best knowledge, no other work has considered and solved these limitations, although they are serious and fundamental problems in the study of object-oriented methods. Our discovery can draw researchers' attention to these problems, and let people recognise

that it is worthwhile providing a good approach to support the study and understanding of the methods, in particular at a time when the methods are so varied and still evolving.

Contribution 1.2 *Establishment of a framework for assessing object-oriented analysis methods*

The *second* contribution of our work is the establishment of a framework, as defined in Chapter 3, to address these problems. This framework provides a much better way of understanding the methods in comparison with the current comparative studies. As discussed in the previous section, and Chapter 1, these comparative studies do not provide a framework like ours and do not cover all essential features found by us when using the methods. In addition, the studies described in Section 1.2 are not based upon experience of using the methods in analysing an application. Instead, the selection of comparative features is based upon the description of the methods, and particularly the favourite of the authors. There is no evidence that these features are helpful in the understanding of object-oriented analysis methods. Therefore, it is dangerous for people to rely on such comparative features for understanding a method. In contrast, our framework was built upon a series of studies and research on a large number of object-oriented methods, including the application of four representative methods as described in Chapter 2. The parts and components of this framework are based upon their roles and relationships in the methods. Details of the contributions of this framework will be described in (2) below.

Contribution 1.3 *Provision of the process and criteria of using the framework*

The *third* contribution of our work is the provision of the process and criteria (as specified in section 3.5) to be used with our framework in helping assess an object-

oriented analysis method. Given this process and criteria, any person can use the framework correctly to identify the essential features of an object-oriented analysis method, and clarify and assess the method to provide a better understanding. Any object-oriented analysis method can be analysed and examined by following the steps of the process and using the criteria provided for each action. We consider that the process and criteria are very useful in the study of object-oriented analysis methods, since they provide a detailed mechanism for using the framework and so make assessment easy and manageable. Unfortunately, current comparative studies do not offer a similar process, and so people are likely to have a problem in using their approaches for similar assessments, particularly when a method is big and complicated. We think that our provision of the process and criteria makes a fundamental improvement in this area.

Contribution 1.4 *Discovery of fundamental assumptions and features about object-oriented analysis*

The *fourth* contribution of our work is the discovery of a set of generic and basic assumptions about object-oriented analysis, and the essential features that occur in the majority of methods, as listed in Table 3.1. We consider that this discovery is new to this area because no other work claimed and showed similar assumptions and features in their publication. For example, the work presented in [Manarchi92] emphasises the importance of using essential features to compare and evaluate a method. However, it does not show which features must be the essential features of object-oriented methods; instead it simply uses the features of traditional methods discovered by Colter [Colter84] and Pressman [Pressman87]. The assumptions and features discovered by us should contribute more to the understanding of object-oriented analysis methods than

the above work, because the assumptions and features of our framework are all abstracted from object-oriented methods rather than traditional methods and so they should reflect better the nature of these methods. We think that this discovery is significant for research into object-oriented methods, since understanding these fundamental assumptions and features means that the nature of individual methods (i.e., what and how each method contributes to object-oriented analysis and so which basic constructs should be included in the method) is understood better. Based on the features, the foundations of an object-oriented analysis method can be appreciated and understood correctly and precisely, as shown by our assessments given by Chapter 4 and 5 and the Appendix. In addition, this discovery can also benefit other research projects, since it provides important information about the foundations of object-oriented analysis methods. Any project can utilise our discovery and reuse these features in their own approaches, if the project relates to the foundations of object-oriented analysis methods and has to cover similar assumptions and features. This reuse of the essential features found can reduce both time and cost of a project.

Contribution 1.5 *Provision of useful experience and results*

The *fifth* contribution of our work is the provision of rich experience of using object-oriented analysis methods (i.e., the methods of [Coad91a, Rumbaugh91, Booch91, Wirfs90]) applied to the same example scenario [Liang93 and 94], and assessing the methods such as Syntropy [Cook94a] in terms of the framework. Such experience and results is useful in observing these methods from different perspectives, for instance, what happens when using different methods for the same application and what methods look similar behind the stated claims. Therefore, such an observation can enhance fundamentally the understanding of the methods.

(2) Specific Contributions of the Framework

Ten object-oriented analysis methods have been assessed as applications of the framework defined in Chapter 3, as described in Chapters 4 and 5 and the Appendix. These assessments show that the content and foundation of a method becomes more and more clear when the essential features are identified from the method and the meanings of these features are clarified by use of the framework. These results also show that the assessment of an object-oriented analysis method by means of the framework is objective in the sense that the account for a given method can always be justified in terms of either what the method claims, or what is actually involved in using the method. This means the method almost writes its own account, if the process and criteria for the framework are followed. In the absence of such a process, other work in this area does not appear to achieve the same objectivity as our work. Specific contributions of the framework, in particular for understanding object-oriented analysis methods, are summarised below.

(a) Major Contributions of the Framework

Contribution 2.1: *Overcome the limitations of current research on object-oriented analysis methods*

As noted before, current comparative studies have failed in helping understand an object-oriented analysis method because of the limitations discussed above. To solve this problem, our framework provides an approach to help understanding by focusing on the essential features of a method, such as 'fundamental concepts' and 'process of analysis'. The framework also emphasises the logical connections between these features (e.g., the relationship between 'fundamental concept' and 'model'). To support better understanding of an object-oriented analysis method, the framework provides a

way to identify and analyse essential features and, more importantly, to examine and clarify the meaning of each feature as it relates to what the method aims to do and how the method supports analysis. We consider that this framework is the first framework that overcomes the above limitations and provides an approach to focusing on understanding rather than only comparing methods.

Contribution 2.2: *Emphasis and assessment of the essential features of object-oriented analysis methods*

As discussed in Chapter 2 and 3, the essential features of the framework actually correspond to fundamental assumptions about object-oriented analysis. The assessment of these features in terms of the framework can be used to explore the fundamental assumptions which are the foundations of object-oriented analysis methods. Understanding the nature of object-oriented analysis methods can be based on this framework. We believe that this is a new contribution to research on object-oriented methods, since the works of other people (such as the comparative studies described in Section 1.2) do not consider what the foundations of methods should be, nor their importance in the methods. In addition, the features considered by other work do not include all the features of our framework, particularly those which we found to be fundamental to analysis in both our use and assessment of methods. For example, Arnold's work [Arnold91] does not consider the feature 'fundamental principle'; Fowler's work [Fowler91] does not consider the feature 'process'; Champeau's work [de Champeau92] does not include the feature 'fundamental concept'; and Monarchi's work [Monarchi92] did not include the feature 'notation'. These methods cannot help people to discover and understand the foundations of a method. This means that our

framework is unique, in that it provides the foundations of object-oriented analysis methods, and can help to understand the nature of the methods.

Contribution 2.3: *Provision of a comprehensive basis for understanding object-oriented analysis methods*

Using the process and criteria provided by section 3.5., the framework enables people to decompose a method into smaller parts, represented as the features illustrated in Table 3.1. For example, the assessments described in Chapter 4 and 5 decomposed five methods into constituent parts, such that the understanding of each method was the aggregate of the understanding of individual parts combined with their relationships in the method. Our experience of using the framework in understanding five methods shows that the process and criteria provide a comprehensive basis for decomposing a method into the constructs and for identifying features. Such a basis makes a method easy to learn and understand.

Contribution 2.4: *Provision of an objective and systematic way to understand object-oriented analysis methods*

There are two disadvantages in the comparative studies described in Section 1.2. The first disadvantage is that they do not establish the extent to which a feature of a method may be apparent but not real. Therefore, assessment of a feature based upon such a study could be subjective rather than objective. The second disadvantage is that they do not provide any process to support the identification of the features of a method. The lack of a process may cause problems, particularly in studying a big and complicated method. However, our framework can establish the essential features of a method, and then find to what extent these features are real rather than apparent by assessing the content and meaning of each feature. To support this, Section 3.5 specifies a well-

defined process of identifying and assessing features, including criteria for each action in the process. Therefore, we consider that our framework provides an objective and systematic way to understand methods, as shown in Chapters 4 and 5. Thus the framework improves the understanding of methods and, more importantly, supports an assessment of what these methods actually take into account during object-oriented analysis, based upon a common infrastructure of the methods.

Contribution 2.5: *A mechanism of assessing the terms and their exact meanings in object-oriented analysis methods*

Researchers have found that many object-oriented analysis methods often use multiple terms for the same concept, or the same term with different meanings may be used in different methods[Synder93]. However, present comparative studies (e.g., the studies shown in Section 1.2) do not provide a mechanism for clarifying terminology definitions, nor the concept which each term represents in a method. In contrast to the comparative studies, our framework provides a mechanism for identifying the definition of terminologies (see Section 3.5.1.1), and then assessing which component each term stands for in this method (e.g., it means a principle, or a concept, or an element of a model) and what is its actual meaning in the method (e.g., data abstraction, or physical entity in the real word, or an encapsulation of data and processes in a system) (see Section 3.5.2.1). This mechanism enables people to consider and distinguish between the defined terms and to find their actual meanings in the method. For instance, in terms of this mechanism, Chapter 4 and 5 showed that the OOA method [Coad91a] and the Booch method [Booch91] define the same term 'object' with different meanings in their object models, and different terms 'encapsulation' and 'information hiding' have the same meaning in the OMT method [Rumbaugh91], the OOA method [Coad91a]

but not in the Wirfs-Brock method [Wirfs90]. Thus this mechanism enables us to recognise and comprehend the actual meaning of terms in a method.

Contribution 2.6: *Identification and Understanding of Implicit Features in object-oriented analysis methods*

Our experience of studying and using object-oriented methods showed that some essential features might not be specified explicitly in a method. For example, the Syntropy method [Cook94a] does not state explicitly the process of analysis in the textbook. Instead the process is implied in the description of 'conceptual model' and 'specification model'. People may not be able to recognise or know how to use implicit features when they are not familiar with a method. In such a situation, the method cannot be understood fully. In order to overcome this shortcoming of the description of a method, a way must be provided to help people recognise and capture implicit features. Unfortunately, no comparative studies of methods had provided such support before our framework was built, since other approaches did not address this issue at all. Only our framework provides support to identify such implicit features, by assessing the logical relationships among explicit features (see Section 3.5.2). For example, Chapters 4 and 5 showed that the implicit process of analysis in the Syntropy method can be identified and determined in terms of the framework, based upon the relationship between the process and the models (i.e., conceptual model and specification model) according to the structure shown by Figure 3.6 and Activity 2.3 described in Section 3.5.2.3.

Contribution 2.7: *Support of understanding strengths and weaknesses of object-oriented analysis methods*

Compared with the other studies described in Section 1.2, our framework is better at identifying and evaluating the strengths and weaknesses of object-oriented analysis methods, because it covers essential features more than other studies do and, more importantly, it also examines the relationships among the features. Thus the features of a method are understood and evaluated individually and in groups according to their relationships. The strengths and weaknesses of methods identified by using the framework should be more objective and accurate than by using other forms of assessment. A proper evaluation of the strengths and weaknesses of object-oriented analysis methods is very important in order to determine which method is the most appropriate one for a specific application.

(b) **Other Contributions of the Framework**

Apart from the above contributions, we consider that our framework can also contribute to the following areas of object-oriented analysis methods:

(i) *Comparison of different object-oriented analysis methods.*

Similar to the comparative studies, our framework can also support comparison of different object-oriented analysis methods, based on the information about the methods expressed in terms of the framework. For example, by comparing the results shown by Chapter 4 and 5, we can find that the type of model is different in the OOA method [Coad91a] and the OMT method [Rumbaugh91]: one type of model is built by the former, and three types of models are constructed by the latter. Therefore, at least there is one significant difference between these two methods. A comparison using the framework can be more objective and accurate than the comparison made by other comparative studies, since the framework can help to obtain extra information (e.g.

relationships between features, extra implicit features) from methods and then takes them into account in the comparison.

(ii) *Comparison of object-oriented analysis methods and traditional methods*

The framework can be used to distinguish between the features of object-oriented analysis methods and the features of traditional methods, where the features are comparable. For example, we can see that the features 'model' and 'process of analysis' also exist in traditional methods, such as 'object model' in the OOA method [Coad91a] and 'data model' in SSADM [Ashworth90]. We can find the difference and similarity between these two models, based upon the contents of the object model identified by using the framework and content of the data model given in the textbook [Ashworth90].

(iii) *Improvement of the existing, or invention of new, object-oriented analysis methods.*

We consider that our framework can be used to develop an object-oriented analysis method. Any existing object-oriented analysis method can be revised and improved based upon the results from an assessment of the method using the framework. Thus a method can be revised by making apparent features real, changing implicit essential features into explicit features, and providing new mechanisms to overcome its weaknesses. For example, Section 5.3.2 showed that few guidelines and criteria were given for the process of analysis in the Booch method [Booch91]. We consider it would be beneficial if more information about this process was provided by a new version of the method. In addition, a new object-oriented analysis method can be created based upon the components and parts of the framework; it requires that

fundamental principles, concepts, models, and so on, must be clearly defined by a new object-oriented method.

(iv) *Education and training of object-oriented analysis methods*

Shelton indicated in his paper[Shelton93] that learning the concepts first is the way to start learning a new object-oriented analysis method. Since no one of the major object-oriented methods is regarded as a standard at present, education and training must show and interpret the detail and nature, in particular fundamental concepts, of every method individually [Fowler93]. We consider that our framework can be used as a reference or training tool for teaching object-oriented analysis methods, since it provides an infrastructure of such methods and it can examine and clarify the meanings and content of the essential features of the methods including fundamental principles and concepts. As a tool, the framework can be used for both training course and self-study:

(a) In a training course, the constructs and structure of a method can be illustrated and interpreted according to the framework, so that students can get a whole picture of the foundation of the method in the end of the course. Based upon this foundation, other aspects of the method, such as a CASE tool, can be further demonstrated and taught.

(b) In a self-study style, people can learn object-oriented analysis methods in terms of the framework. By following the process provided by Section 3.5, people can (i) identify and understand explicit features of the methods according to the framework, and understand the 'what' aspect and 'how' aspect of the method, and (ii) assess the relationships among the features, understand the impact of one feature on another

feature, and identify implicit features. All constructs and structure of methods will become evident through this process.

(v) Development of computer-assisted learning software for object-oriented analysis methods

Computer-Assisted Learning(CAL) is a new approach to education and training, in particular for self-study or distance learning. Currently, teaching and training for object-oriented methods is almost always taught in a classroom context. If we want to teach the methods by computer, we need some form of CAL software for this purpose. We consider that our framework provides a common basis for developing such software, since it provides a generic pattern of the essential features (or constructs) of object-oriented analysis methods, and illustrates the dependency between the features. The framework can be regarded as an infrastructure that can be used for CAL software for teaching and demonstrating various object-oriented analysis methods, and possibly comparing and evaluating the methods. The other works noted in Section 1.2 cannot be used in this way, since they do not cover all essential features included in our framework and, more importantly, do not provide a generic pattern to analyse and organise the features for the methods in a hierarchical structure. Therefore, they cannot play the same role as our framework in the development of CAL software.

6.2 Other Potential Use of the Framework and Future Work

Since our framework includes the essential features of object-oriented analysis methods and also focuses on the impact of one feature on another feature, we consider that the

framework can be regarded as a vehicle for establishing evaluation criteria of the methods, and for developing CASE tools to support computer-assisted object-oriented analysis.

6.2.1 Establishment of Evaluation Criteria

Our framework contains the essential features which a majority of object-oriented analysis methods must support, even though the contents of features may vary for different methods. Thus methods can be compared and evaluated based upon these critical features; in addition, the relationships between the features are emphasised by the framework, and so the framework provides a sound basis for evaluating the features of the methods. Generic evaluation criteria can be established based upon the framework for evaluating the methods accurately. For example, the following evaluation criteria may be considered in the evaluation of methods:

— the definition of each fundamental concept should be consistent with at least one fundamental principle;

— a fundamental concept must be supported by at least one element in a model;

— the notation used by a method must represents all elements in the model(s) clearly, precisely and simply;

— each step/activity of the process of analysis should include detailed guidelines and criteria;

— the products of analysis must be complete and consistent with the system requirements.

6.2.2 Development of Generic CASE Tools for Object-Oriented Analysis

A CASE tool for object-oriented analysis is considered useful in reducing the effort required for, and improving the quality of, the end-product in system development [Sully93]. Although some tools like 'StP/OMT' [IDE93] are now available for a specific method, it would be beneficial to develop generic object-oriented analysis tools for use with different object-oriented analysis methods. For this purpose, the framework can be regarded as a basis for developing such a CASE tool since it shows the common features of object-oriented analysis methods, so that the CASE tool can support object-oriented analysis for a number of the methods.

Appendix

An Outline of the Assessment of Five Other Analysis Methods Using the Framework

In addition to the five methods assessed in Chapter 4 and 5, we have also used the framework for the assessment of five another methods; i.e., the Shlaer and Mellor method [Shlaer88 and 91], OOSE [Jacobson92], OSA [Embley92], Ptech [Martin92a] and HOOD [Robinson92]. This appendix outlines the assessment of these methods, using the framework.

1. The Shlaer/Mellor Method

It has been argued that the early version of the Shlaer and Mellor method [Shlaer88] was not really object-oriented, for reasons such as the lack of provision of a notion of inheritance [Graham94]. However, its later version [Shlaer91] includes inheritance, as well as operations (of objects) by modelling the lifecycles of objects with state transition diagrams. By the method, various analysis models are built, such as the information model, the state model, and so on. The 'what' and 'how' aspects, with the essential features, of

the later version of the method (i.e., [Shlaer91]) are identified and assessed in terms of the framework, as follows.

1.1 The 'What' Aspect of the Shlaer/Mellor Method

The essential features of the Shlaer/Mellor method — and their relationships — are shown in Table 1.1. The assessment shows that the principle 'abstraction' in the method refers to all data abstraction, process abstraction and behaviour abstraction. The fundamental concepts 'object structure', 'inheritance relationship' and 'system function' are found to be implicitly used in the method by examining the meaning of the elements in the 'model' column of the table. The implicit fundamental principles 'information hiding' and 'scale' are also used (implicitly) in the method. The underlying concepts 'object structure' and 'domain' also hold.

By considering the meanings and roles of the essential features that reflect the 'what' aspect of the Shlaer/Mellor method and their relationships, the content of this aspect are — in outline — as follows:

• Fundamental Principles

— *Principles on object orientation*

— *Abstraction on data, process and behaviour*

— *'Inheritance' and 'information hiding' are used.*

• Fundamental Concepts

— *Concepts around objects*

Principle / Part / Stage	Fundamental Principle	Fundamental Concept	Model	
			Element	Type
Analysis	•Abstraction (data)	•Object •Instance	•Object •Instance	•Information Model
	•(Information hiding)	•(Object structure)	•Attribute •Relationship	
	•Inheritance	•(Inheritance relationship)	•Sub/supertype (Single/ multiple)	
	•Abstraction (behaviour)	•Lifecycle of object	•Object state •Event •Transition rule •Action •Timer •Lifecycle of relationship •Object interaction	•State Model •Object Communication Model
	•Abstraction (process)	•(System function)	•Process •Data store •Data flow •Control flow	•Process Model
	•(Scale)	•Domain	•Subsystem	•Domain-level Models

Table 1.1 The Relationship between the Features in the 'What' Aspect of the Shlaer/Mellor Method

— *The concept 'object' means a single typical but unspecified instance of something in the real world*

— *Focus on object structure and object life cycle*

—*The concept 'inheritance relationship' is implicitly supported*

— *'Aggregation' is not emphasized in particular, instead the object structure is more concerned with the concept 'association' between objects.*

• Object Model

— *The element 'object' includes attributes, state (values of the attributes), and actions (processes).*

— *Emphasis on data abstraction by regarding the object structure as a basis of other models*

— *Four specific object-oriented models are built by this method*

— *Strong focus of subsystem modelling in analysis*

— *Aggregation is not emphasized*

— *The life cycle of relationships is also modelled*

— *The element 'timer' is a mechanism that can be used by an action to generate an event at some time in future*

— *The processes in the process model depend on the actions in the state models*

— *Focus on the relationships, communications and accesses between subsystems.*

1.2 The 'How' Aspect of the Shlaer/Mellor Method

The essential features that reflect the 'how' aspect of the Shlaer/Mellor method are identified and listed in Table 1.2, in terms of the practice part of the framework. The relationship between the features in the 'how' aspect of the method is described in Figure 1.1.

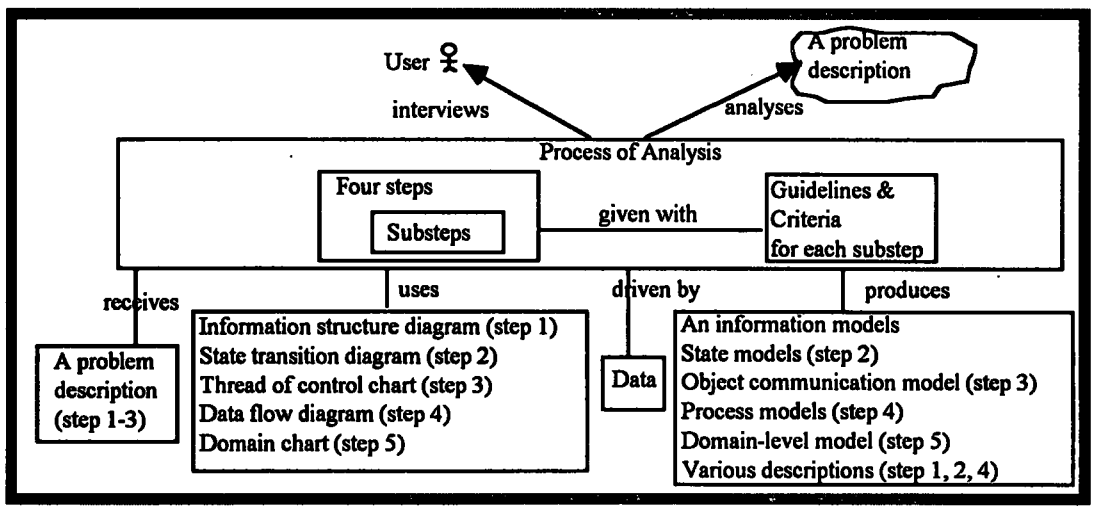

Figure 1.1 The Relationship between the Features in the 'How' Aspect of the Shlaer/Mellor method

Practice Part / Stage	Notation	Tactic of Analysis	Input	Process of Analysis: Step	Process of Analysis: Substep	Process of Analysis: Guideline & Criterion	Product
Analysis	•Information Structure Diagram •State Transition Diagram •Thread of Control Chart •Data Flow Diagram •Domain Chart	Analyse the problem description by •(Data-driven)	•A problem description •dialogue with users	•Constructing the information model	•Identify objects •Identify attributes •Identify associations •Define subtype and supertype	•Consider the conceptual entities, ... •One value for each attribute at any ..... •Consider conditional or unconditional relation..... •Consider the objects having common attributes...	•An information model •Object descriptions •Relationship descriptions •State models •Action descriptions •An event list •An object communication model •Process models •Process descriptions •Object access model •Domain-level models —A subsystem relationship model —A subsystem communication model —A subsystem access model
				•Constructing state models	•Identify states for each object •Identify events •Define actions •Define transition rules •Define timers •Form object lifecycles •Form sub/supertype lifecycles	•Abstract common behaviour of each object..... •Abstract events from incidents from problem..... •Actions can be exeuted by any instance of object... •Consider new state entry,..... •Consider the expiration of a time interval ..... •Consider the purpose and context of each state..... •When an instance of the supertype object migrates..	
				•Constructing the object communication model	•Define dynamics of relationships between objects •Build layers of communication •Draw a control thread chart	•Consider creation and deletion of relationships •Consider how each relationship evolves over time... •Place objects without details in the lowest layer •Place object with details in higher layer,..... •Consider the sequence of actions occurring in	
				•Constructing the process model	•Specify actions •Draw action data flow diagrams •Partition each ac basic processes •Draw control flows •Draw an object access diagram	•The problem is first partitioned into objects, then into actions, and finally into processes, ... •A data flow is drawn if data is produced by one process and consumed by another, ... •Decide constraints on the oder of processes ..... •Consider the objects that interact with one another both through events or accessor processes	
				•Constructing domain-levels models	•Define domains •Build subsystem relationship model •Build subsystem communication model •Build subsystem access model	•Identify the domains needed for the system ..... •Describe any inter-subsystem relationships by a line connecting the two subsystems, ... •Draw an arrow from the subsystem in which the event is generated to the subsystem receiving the event ... •Draw an arrow from subsystem A to subsystem B if a state model in A accesses to an object in B.....	

Table 1.2 The Essential Features in the 'How' Aspect of the Shlaer/Mellor Method

1.3 The Relationships between the Two Aspects of the Shlaer/Mellor Method

(1) Analysis Models and Tactic of Analysis

The information model is the basic model built by this method. A data-driven process of analysis is therefore provided by the method.

(2) Analysis Models and Process of Analysis

— *Five major models built by four steps with five substeps*

— *Each element in each model defined by the substeps of a step*

— *An information model is always constructed first since it is a basic model*

(3) Analysis Models and Notation

In order to represent five models, five notations are used in the Shlaer/Mellor method. The symbols used to represent an information model and a domain-level models are illustrated in Figure 1.2. The symbols used to represent a state model and an object communication model are shown in Figure 1.3. The process model is represented by data flow diagrams, in the same fashion as shown in Figure 4.6 given in Chapter 4.

(4) Analysis Models and Products of Analysis

The products of analysis in the Shlaer/Mellor method consists principally of five models, as shown in Table 1.2.

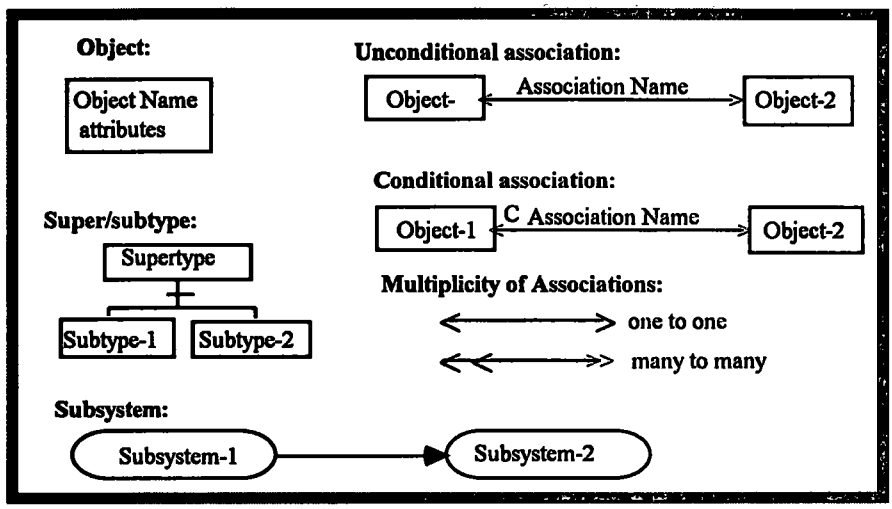

Figure 1.2 Notation of Information Model and Domain-Level Model

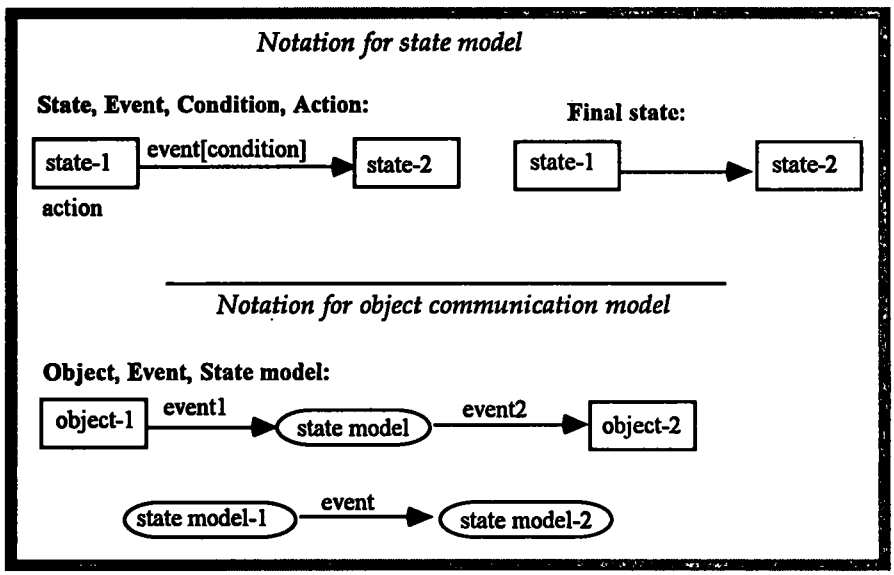

Figure 1.3 Notation of State Model and Object Communication Model

2. OOSE (Jacobson *et al.*)

OOSE (Object-Oriented Software Engineering) [Jacobson92] is an object-oriented method for analysis, construction (including design and implementation), and testing in software development. OOSE is derived from Jacobson's Objectory (the Object Factory for Software Development) [Jacobson87] and it is a simplified version of Objectory. It in

particular emphasises and provides a use case driven process of analysing and designing software systems. A use case model is included in the requirement model to describe the things (actors) existing outside a system and the processes (use cases) performed by the system. The 'what' and 'how' aspects with the essential features of OOSE are identified and assessed by means of the framework.

2.1 The 'What' Aspect of OOSE

The essential features of OOSE and their relationships are shown in Table 2.1. The principle 'abstraction' of OOSE refers to 'data abstraction' and 'process abstraction' as OOSE focuses on analysing and modelling the data and process aspects of a system.

Principle / Part / Stage	Fundamental Principle	Fundamental Concept	Model	
			Element	Type
Analysis	•Abstraction (data)	•Object •Class	•Object •Class	•Requirement Model —Problem domain model —Use case model
	•Encapsulation	•Object structure	•Attribute •Operation •Actor •Use case	
	•Information hiding •Abstraction (process)		•Entity object •Interface object •Control object •Association	•Analysis Model
	•Inheritance	•Inheritance relationship	•Gen-Spec Relationship (Single/ multiple)	
	•Scale	•Grouping	•Subsystem •DependsOn relationship	

Table 2.1 Relationship between the Essential Features in the 'What' Aspect of OOSE

The content of this aspect is outlined below, after considering the meanings and roles of the essential features and their dependency in OOSE for analysis:

• Fundamental Principles

— *Principles of object orientation*

— *Abstraction with data and process*

— *Support of 'inheritance' and 'encapsulation' and 'information hiding'. In particular, 'encapsulation' and 'information hiding' are emphasised separately, as in the Wirfs-Brock method.*

— *Support of the scale of systems*

• Fundamental Concepts

— *Concepts around objects*

— *The concept 'object' means people and entities in the real world.*

— *Focus on object structure but not object behaviour*

— *Modelling 'use connection' between objects by 'object structure'.*

• Models

— *Two models are built: a requirement model that captures the functionality requirements, and an analysis model that specifies all the logical objects to be included in a system and their relationships and subsystems. The analysis model forms a basis for the system's structure.*

— *The requirement model helps to keep track of the requirements of a system right through its whole life cycle. The analysis model supports inheritance relationships and associations between objects including the description of aggregation.*

— *The element 'object' is characterised by a number of operations and a state which remembers the effect of these operations on the object.*

— *Objects are classified into three types of objects in the analysis model: entity objects capturing the information in a problem domain which is stable within a system; interface objects modelling behaviour and information dependent on the interface to the system; and control objects specifying behaviour which operates on several different entity objects, returning the result to interface objects.*

— *Support for data abstraction and process abstraction by modelling object structure with attributes, operations, and use cases.*

— *Description of interactions between users and a system in terms of the elements 'actor' and 'use case'. The actors represent what interacts with a system. A use case is a sequence of transactions in a dialogue between a user and the system.*

— *Objects are grouped into subsystems in an analysis model.*

— *The state changes inside an object are not particularly described.*

2.2 The 'How' Aspect of OOSE

The essential features that reflect the 'how' aspect of OOSE are identified by using the framework and illustrated in Table 2.2. The features show that OOSE supports a use case driven (i.e., process-driven) tactic of analysis, by which the process of analysis constructs a requirement model and an analysis model sequentially. The relationships between the features in the 'how' aspect of the method are further considered as shown in Figure 2.1.

Practice / Part / Stage	Notation	Tactic of Analysis	Input	Process of Analysis			Product
				Step	Actions	Guideline & Criterion	
Analysis	•OOSE notation	Analyse a requirement specification by •Use case	•Requirements specification •Dialogue with users	•Building the requirement model	•Identifying actors and use cases	•An actor is a user who interacts with the system... •Ask the questions such as "will the actor have to read/write/change any of the system?", ...	•A requirement model —a use case model —user interface —a problem domain model •An analysis model •Descriptions of objects and classes
					•Describing user interfaces		
					•Identifying problem domain objects	•Capture all the important concepts in a problem domain, consider similarities of objects	
				•Building the analysis model	•Finding objects	• Consider about nouns; learn terminology in problem domain .... •Select those objects relevant to the system •Use different rules for defining different objects	
					•Organising objects	•Consider similarities of classes in order to define Gen-Spec relationships •Consider dependency between objects, •Consider how an object is a part of another •Group objects into subsystems	
					•Identifying object interaction	•Consider different scenarios or use cases •Decide object's interface from these scenarios •Consider how certain objects are part of other objects	
					•Identifying operations on objects	•Consider an object's interface •Consider the application •Divide a complex object into simpler objects	
					•Identifying attributes of objects	•Consider the information which each object holds	

Table 2.2 The Essential Features in the 'How' Aspect of OOSE

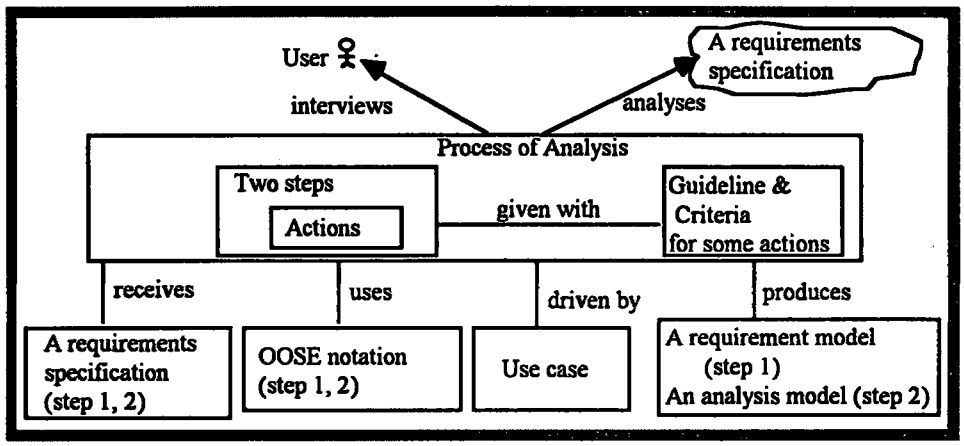

Figure 2.1 The Relationship between the Features in the 'How' Aspect of OOSE

2.3 The Relationships between the Two Aspects of OOSE

(1) Models and Tactic of Analysis

All the requirement and analysis models describe objects and their structures by focusing on the use cases between objects. A use case driven tactic of analysis is thus offered by OOSE to support such emphasis.

(2) Models and Process of Analysis

— *The requirement and analysis models built by two steps including eight actions.*

— *Actions in the process of analysis support the specification of a system by means of various elements in these two models.*

— *The process of analysis includes the guidelines and criteria which help to build the two models precisely and correctly.*

(3) Models and OOSE Notation

OOSE uses an OOSE notation (see Figure 2.2) which specifically represents a requirement model and an analysis model.

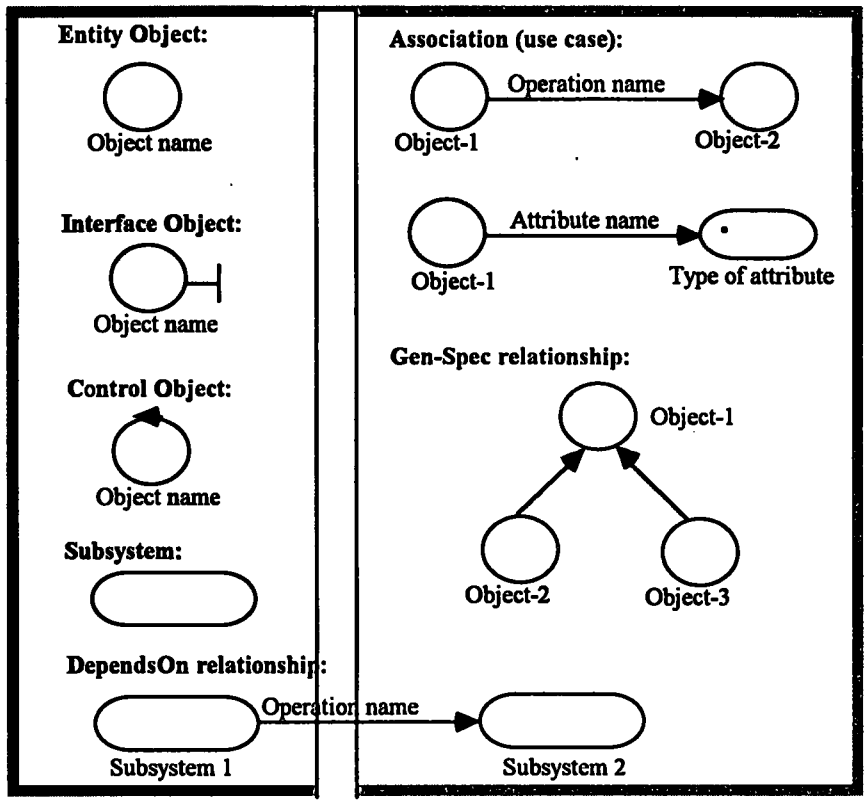

Figure 2.2 OOSE Notation

(4) Models and Products of Analysis

The products of analysis from OOSE include a requirement model and an analysis model, as well as the description of each object in these two models.

3. OSA (Embley et al.)

OSA (Object-Oriented System Analysis) [Embley92] is an object-oriented method which was developed at Hewlett-Packard. OSA focuses on the conventional tripartite division of analysis into three separated but related activities with a specific notation for each. The 'what' and 'how' aspects with the essential features of OSA are identified and then assessed in terms of the framework.

3.1 The 'What' Aspect of OSA

In OSA, a system is defined as a group of objects. The analysis thus focuses on the study of a specific domain of interacting objects for the purpose of understanding and documenting their essential characteristics, as understanding a system before design is critical to the success of creating a complex software system. Such a understanding is achieved by building a concept model of the system. The essential features of OSA, and their dependencies, are shown in Table 3.1. In the table, the concepts 'object', 'object relationship' and 'object behaviour' emphasise the static structure and dynamic behaviour of objects within a system. These concepts refer to the principle 'abstraction', being abstraction of data and behaviour. Another two implicit principles — 'encapsulation' and 'information hiding' — are derived from the meanings of the fundamental concepts 'object' (consisting of characteristics and behaviour) and 'view of system' (hiding different levels details) addressed by OSA.

According to the meanings and roles of the essential features in the 'what' aspect of OSA, as well as the relationships shown in Table 3.1, the content of this aspect are outlined as follows:

• Fundamental Principles

— *Principles of object orientation*

— *Abstraction with data and behaviour*

— *'Encapsulation', 'inheritance' and 'information hiding' are used.*

Principle / Part / Stage	Fundamental Principle	Fundamental Concept	Model	
			Element	Type
Analysis	•Abstraction (data)	•Object •Classification	•Object •Object class	•Object-relationship Model
	• (Encapsulation)	•Object relationship	•Relationship •Association •Aggregation •Constraint	
	•Inheritance	•Inheritance relationship	•Generation-Specification (Single/ multiple)	
	•Abstraction (behaviour)	•Object behaviour	•Object state •Event •Action •Transition •Constraint •Object interaction	•Object-behaviour Model •Object-interaction Model
	• (Information hiding)	•View of system	•High-level view •Exploded view •Imploded view	

Table 3.1 Relationship between the Features in the 'What' Aspect of OSA

• Fundamental Concepts

— *Concepts around objects.*

— *The concept 'object' means a person, place, or thing in reality.*

— *Focus of object relationships and object behaviour.*

—*The concept 'inheritance relationship' is supported.*

— *A concept 'view of system' is used to manage the complexity of a system.*

• Models

— *The element 'object' includes states and actions (operations).*

— *'Relationship' means a logical connection among objects; 'association' means a member of a relationship; and 'aggregation' means being part of a relationship.*

— *Emphasis on the object relationships by modelling various object relationships and on the object behaviour by modelling object states, state transitions and interactions.*

— *Does not explicitly represent the attributes of each object class (i.e., class and its objects) in a system.*

— *Object-behaviour model describes the behaviour of each object in a system by representing its perceived states, conditions and the events affecting it, and the actions performed by or on it.*

— *Events can be modelled as objects.*

— *Three specific types of model are built by OSA.*

— *Provision of different level views of a system.*

3.2 The 'How' Aspect of OSA

The essential features that reflect the 'how' aspect of OSA are listed in Table 3.2. It shows that this method supports a model-driven tactic of analysis and the process of analysis aims to build three different types of model in high-level views of a system. The relationship between the features in this aspect of the method is shown in Figure 3.1.

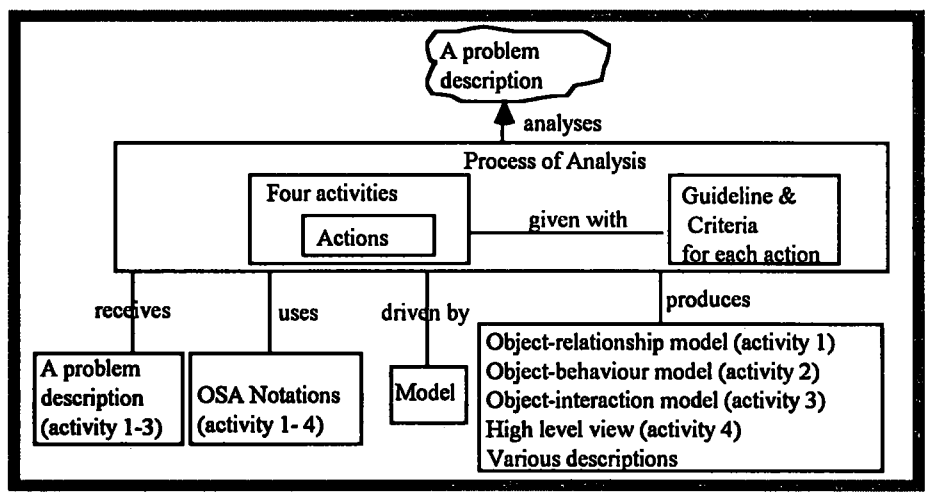

Figure 3.1 The Relationship between the Features in the 'How' Aspect of OSA

Practice / Part / Stage	Notation	Tactic of Analysis	Input	Process of Analysis			Product
				Activity	Action	Guideline & Criterion	
Analysis	•Notation for object-relationship model •Notation for object-behaviour model •Notation for object-interaction model •Notation for levelling	Analyse the system requirement by •Model-driven	•A problem description •dialogue with users	•Building the object-relationship model	•Identify objects •Identify relationships •Define Gen-Spec. •Define aggregation •Define association	•Nouns and noun phrases, etc. •Verbs and verb phrases, etc., consider constraints... •Consider common behaviour of object classes •Consider subparts of objects •Consider the members associated to form an object	•An object-relationship model •An object-behaviour model •An object-interaction model •High-level views of object models •Descriptions of object classes, interactions, etc.
				•Building the object-behaviour model	•Identify states •Define triggers and transitions •Draw state nets	•Observe an object in a system..... •Consider expected events, system conditions triggering a change of state for an object, actions,..... •Integrate all states for all objects into an object class	
				•Building the object-interaction model	•Draw basic interaction diagrams •Describe interactions •Define interaction activities •Define high-level interaction	•Look at the interactions between object classes, examine the state nets and communications..... •Consider the details of interactions between objects •Consider "access", "modify", "remove", "destroy", "add", and "create" activities, constraints •Abstract states by omitting some of details, group interactions together	
				•Constructing high-level views	•Construct high-level views of the object-relationship model •Define high-level object classes •Define high-level states	•Create high-level relationship sets, ensure validity when including an object class in the creation of a high-level relationship set, ... •Subsume a class name chosen independently or the name of one of object class in the high-level class •Consider transition at various levels of abstraction, choose an appropriate level of abstraction	

Table 3.2 The Essential Features in the 'How' Aspect of OSA

3.3 The Relationships between the Two Aspects of OSA

(1) Models and Tactic of Analysis

Three models are built by a model-driven process of analysis in OSA. These models are constructed in any sequence in OOA.

(2) Models and Process of Analysis

— *Three types of model are built in four activities with a set of actions.*

— *Each element in each model is defined in individual actions.*

— *High-level views of a system are constructed by a specific activity.*

(3) Models and OSA Notation

The three object models in OSA are represented by three specific notations, as shown in Figures 3.2-4. Another notation (see Figure 3.5) is also used in order to illustrate the different levels of viewing these three models.

(4) Models and Products of Analysis

The products of analysis from OSA comprises three types of object model as well as the descriptions of object classes and interactions among them, as shown in Table 3.2.

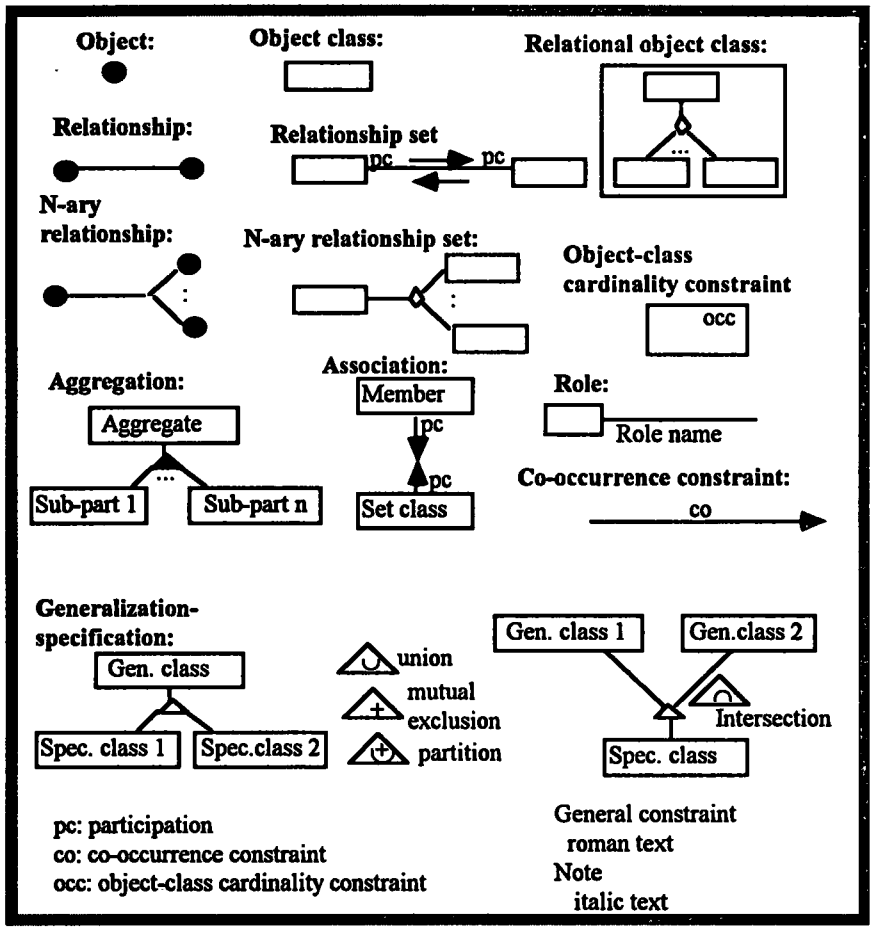

Figure 3.2 Notation for the Object Relationship Model

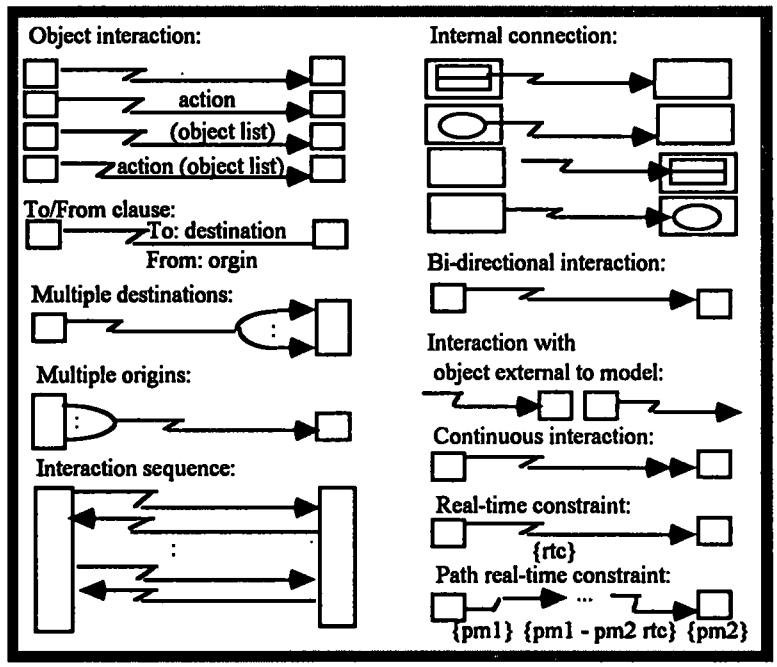

Figure 3.3 Notation for the Object Interaction Model

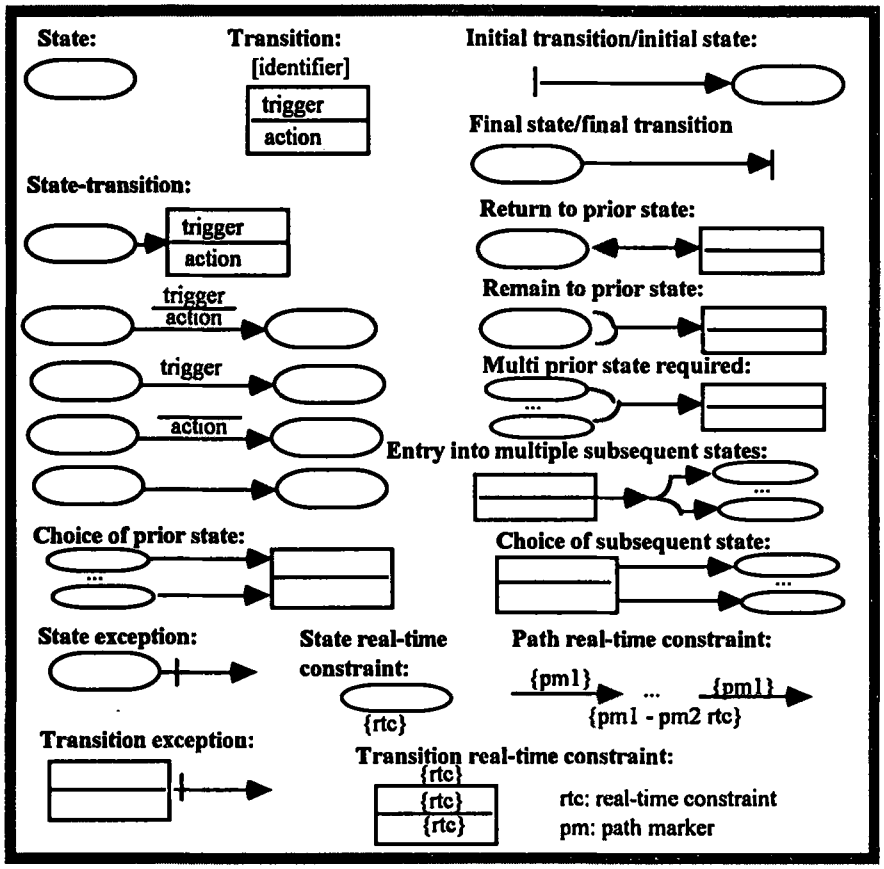

Figure 3.4 Notation for the Object Behaviour Model

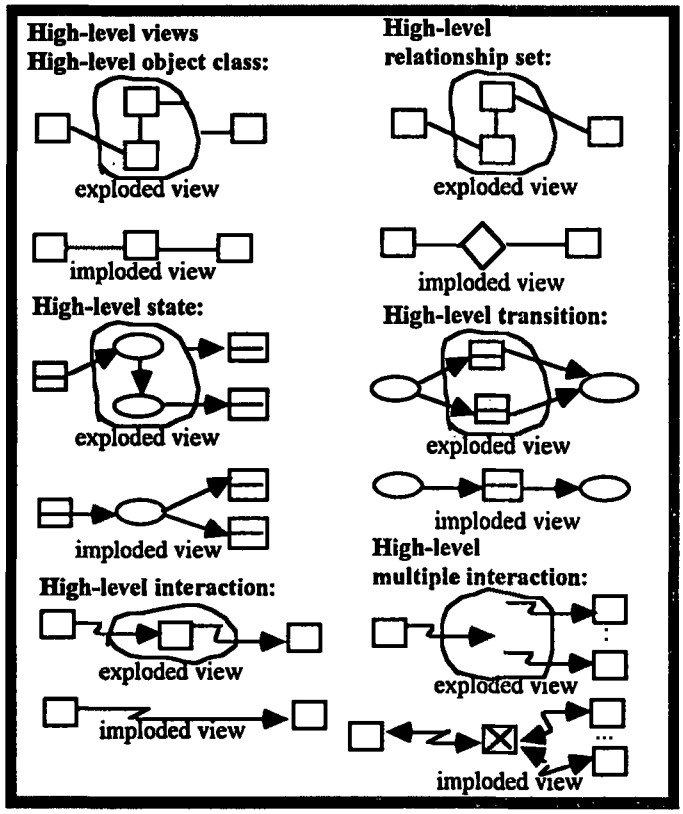

Figure 3.5 Notation for Levelling in OSA

4. Ptech (Martin and Odell)

Ptech is described in the book by Martin and Odell [Martin92a]. It is claimed that this method supports object-oriented analysis and design. The assessment of Ptech in outline is as follows.

4.1 The 'What' Aspect of Ptech

In Ptech, object-oriented analysis is regarded as a process of modelling the world in terms of objects (which have properties and states) and events triggering operations (which change the state of objects). Thus, the essential features in the 'what' aspect of Ptech emphasise the modelling of objects, object behaviour, etc. as shown in Table 4.1. The principle 'abstraction' is regarded as the abstraction of data, behaviour and process. In addition, the principle 'communication with message' is also implicit in Ptech, with the concept 'message' as a notion of object communication. The element 'composition' in an object structure model should be defined as being underpinned by the implicit concept 'aggregation'.

By examining the meanings and roles of the essential features in the 'what' aspect of Ptech and the relationships between the features, the content of this aspect is, in outline, as follows:

• Fundamental Principles

— *Principles of object orientation*

— *Abstraction with data and behaviour and process*

— *'Encapsulation' also means 'information hiding'. The principles 'inheritance' and 'hierarchy' are regarded as fundamental in this method.*

Principle / Part / Stage	Fundamental Principle	Fundamental Concept	Model	
			Element	Type
Analysis	•Abstraction (data)	•Object •Object type	•Object •Class	•Object structure Model
	•Encapsulation (Information hiding)	•Relationship •(Aggregation)	•Data(Property) •Association •Composition	
	•Inheritance	•Generalisation	•sub/supertype (Single/multiple)	
	•Hierarchy	•Partition	•Subset of objects	
	•Abstraction (behaviour)	•Lifecycle of object	•Object state •Event •Control condition •Operation	•Object behaviour Model
	•(Message coomunication)	•Message	•Message passing	
	•Abstraction (process)	•System function	•Activity •Product •Object flow	

Table 4.1 The relationship between the Essential Features in the 'What' Aspect of Ptech

• Fundamental Concepts

— *Concepts around objects.*

— *The concept 'object' means any real or abstract thing about which we store data and the methods which manipulate the data.*

— *Focus of object structure and object life cycle and system function.*

— *The concept 'generalisation' is defined to mean an inheritance relationship.*

— *Support for 'aggregation'.*

• Models

— *The element 'object' includes properties (data types) and the permissible operations acting on the properties. An object may be composed of other objects.*

— *Emphasis on object structure modelling which forms a basis for object behaviour modelling.*

— *Modelling object behaviour including drawing object-flow diagrams that provide high-level views of processes in a system and indicate the objects flowing among the processes.*

— *Construction of two specific models.*

— *Different levels of object structure and behaviour are represented by subsets of objects and states.*

— *Support for modelling object interactions by 'message passing'.*

4.2 The 'How' Aspect of Ptech

The essential features in the 'how' aspect of Ptech are recorded in Table 4.2. The relationships between these features are shown in Figure 4.1.

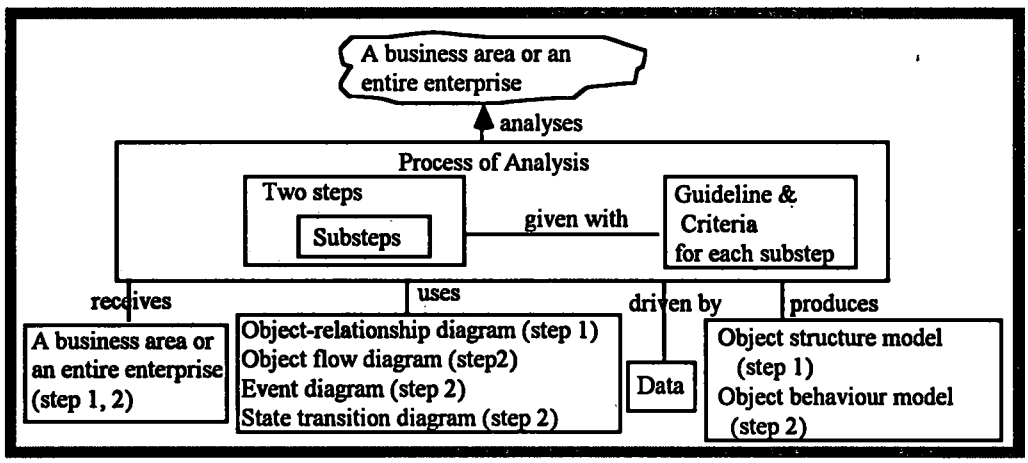

Figure 4.1 The Relationship between the Features in the 'How' Aspect of Ptech

Practice Part / Stage	Notation	Tactic of Analysis	Input	Process of Analysis			Product
				Step	Substep	Guideline & Criterion	
Analysis	•Object-relationship diagram •Object-flow diagram •State transition diagram	Analyse a business or an entire enterprise by •(Data-driven)	•A business area or an entire enterprise	•Object structure analysis	•Identify objects and their associations •identify sub/supertype •Identify composed-of structure	•Consider what types of objects exist •Consider what the relations and functions of objects are •Consider what subtypes and supertypes are useful •Consider the objects that can be composed of other objects	•Object structure model •Object behaviour model
				•Object behaviour analysis	•Clarify event type •Generalise event type •Define operation conditiond •Identify operation causes •Refine cycle results •Analyse complicated processes in a problem •Identify products from the processes •Draw an object-flow diagram	•Specify event prestates and poststates in terms of object types •Consider the low-level event types •Define the operations with conditions for each object type in the object structure model •Define trigger rules for events. •Generalise, specialise or even remove object types further •Determine the essential activities managing the operations and resources of an organisation •Consider the purpose of the activities •Describe key enterprise activities linked by the products which activities produce and exchange	

Table 4.2 The Essential Features in the 'How' Aspect of Ptech

4.3 The Relationships between the Two Aspects of Ptech

(1) Models and Tactic of Analysis

An object structure model is the basic model in Ptech. A data-driven tactic of analysis is therefore used by the method.

(2) Models and Process of Analysis

— *Two type of object model built by two steps including a set of substeps*

— *The object behaviour model that is built includes the process view of a system (i.e., object-flow diagram)*

— *Each element in a model is described by an individual substep.*

(3) Models and Notation

In order to represent an object structure model and an object behaviour model, a notation (e.g., object-relationship diagram) is offered by Ptech, as shown in Figures 4.2-4.

(4) Models and Products of Analysis

The products of analysis produced by Ptech comprise an object structure model and an object behaviour model.

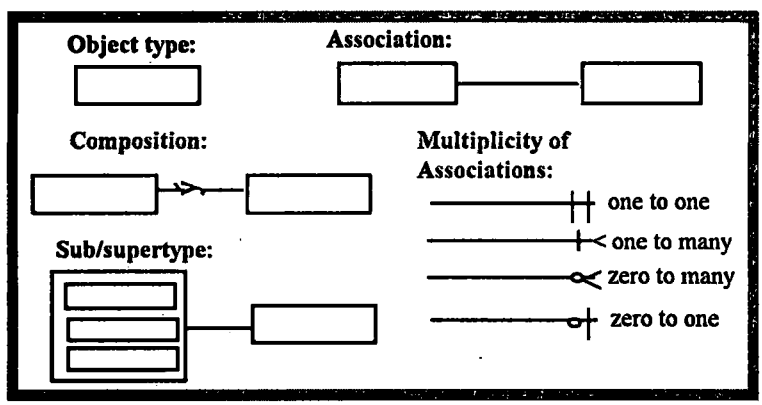

Figure 4.2 Symbols in Object-Relationship Diagram

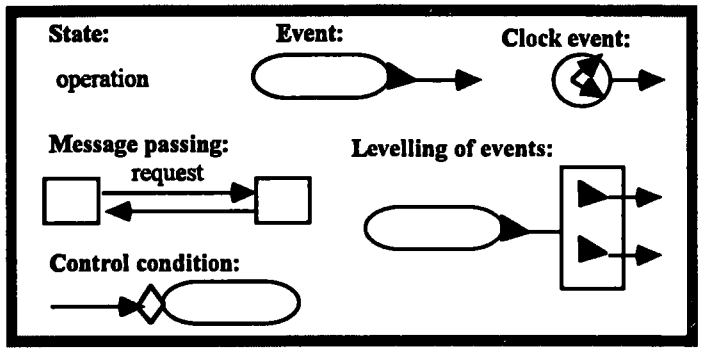

Figure 4.3 Symbols in Event Diagram

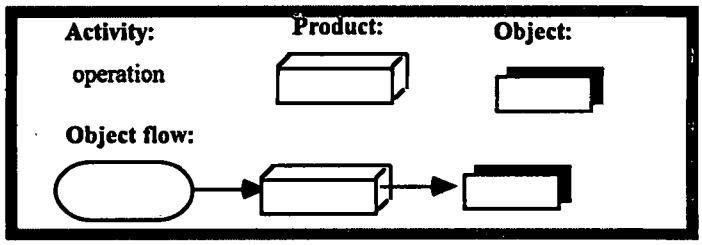

Figure 4.4 Symbols in Object-Flow Diagram

5. HOOD

HOOD (Hierarchical Object-Oriented Design) [Robinson92] is an object-oriented method that was specially developed for designing software to be written in Ada. Although HOOD focuses on architectural design in software development, it also does object-oriented analysis for identifying objects from system requirements which describe a problem. HOOD can therefore be used as an object-oriented analysis method.

5.1 The 'What' Aspect of HOOD

The essential features and their relationships in the 'what' aspect of HOOD are shown in Table 5.1. The assessment of this aspect shows that the fundamental principle 'abstraction' is regarded as process abstraction in HOOD. A fundamental concept 'aggregation' is implicitly used by this method, according to the meanings and roles of the elements 'included relationship' and 'implemented-by relationship' in an object model.

Principle / Part / Stage	Fundamental Principle	Fundamental Concept	Model	
			Element	Type
Analysis	•Abstraction (process)	•Object •Class	•Object (active, passive) •Class object	•Object model
	•Information hiding (encapsulation) •Modularity	•Object control structure •(Aggregation)	•Operation •Data flow •Exception flow •use relationship •Parent/child object •Included relationship •Implemented-by link	

Table 5.1 Relationship between the Essential Features in the 'What' Aspect of HOOD

The main characteristics of these fundamental principles and concepts and the object model recorded in Table 5.1 are outlined as follows:

• Fundamental Principles

— *Principles of object orientation*

— *Abstraction with process*

— *'Information hiding' (also encapsulation) is fundamental in HOOD.*

— *'Hierarchy' is strongly supported by HOOD in modelling.*

• Fundamental Concepts

— *Concepts around objects*

— *The concept 'object' means a model of a real-world entity*

— *Focus on an object control structure that is a notion of co-operations between objects.*

— *'Aggregation' is emphasised in particular as a notion of object assembly.*

• Object Model

— *The element 'object' combines data and operations in such a way that data is implied in the definition of an object and accessed through the operations of the object. An object may have a state. An object may be an attribute of another object, i.e., it is a child object of that object (its parent).*

— *Emphasis on process abstraction by modelling object operations and their 'use relationships'.*

— *One object model is built by this method.*

— *Modelling the inclusion relationships between objects (parent/child objects) in a hierarchical structure.*

— *Inheritance relationship is not modelled by HOOD.*

5.2 The 'How' Aspect of HOOD

The essential features in the 'how' aspect of HOOD are recorded in Table 5.2, and the relationships between these features are shown in Figure 5.1.

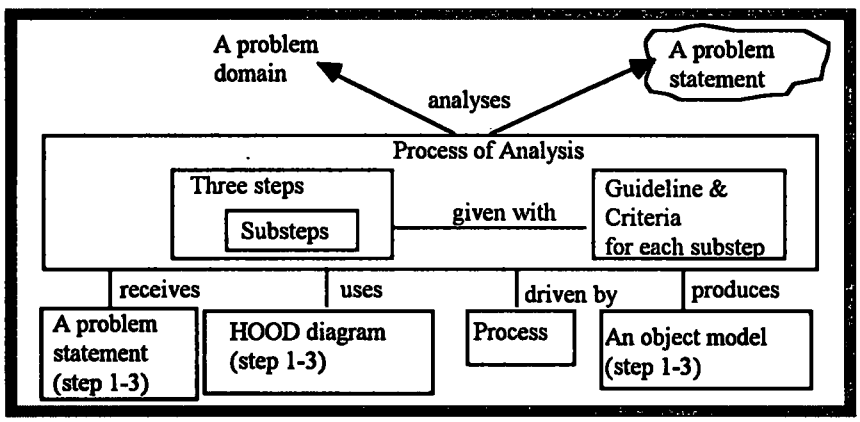

Figure 5.1 The Relationship between the Features in the 'How' Aspect of HOOD

Practice / Part / Stage	Notation	Tactic of Analysis	Input	Process of Analysis			Product
				Step	Substep	Guideline & Criterion	
Analysis	•HOOD diagram	Analyse the problem statement or a problem domain by •(Process-driven)	•The problem statement •A problem domain	•Defining problem	•State the problem •Analyse the requirements	•Provide a clear and precise definition of the problem •Decide the context of the system to design •Gather and analyse all the information relevant to the problem, e.g., determine the static functional requirements and determine the dynamic functional requirements	•An object model
				•Developing an informal solution strategy	•Create an outline solution of the problem •Consider real-world objects and their actions	•Describe the solution in a hierarchical structure by using a natual language •In the top level, the solution should describe the system in terms of real-world objects associated with the actions which may be performed on them •This solution should not be considered as definitive, but as a baseline for further refinement in the following phase	
				•Formalise the strategy	•Identify objects from the solution strategy •Identify operations •Group objects and operations •Draw HOOD diagram	•Abstract nouns and noun phrases •Structure these nouns according to their behaviour •Decompose the nouns until no new object is found •Objects may be parent objects, child, or terminal objects •Each operation is associated with one object •Describe objects with operations in HOOD diagram •Identify child objects by decomposing parent objects •Add corresponding data flows •Define implemented-by relationships	

Table 5.2 The Essential Features in the 'How' Aspect of HOOD

5.3 The Relationships between the Two Aspects of HOOD

(1) Object Model and Tactic of Analysis

An object model is built by HOOD through analysis. Since this model focuses on specifying the operations within objects and the control structure of objects, a process-driven tactic of analysis is thus used in HOOD.

(2) Object Model and Process of Analysis

— *Three steps are included for building an object model.*

— *Each element in a model is described by a substep in the steps.*

(3) Object Model and HOOD Diagram

A HOOD diagram is used by HOOD to represent an object model. The symbols in the diagram are illustrated in Figure 5.2.

(4) Object Model and Products of Analysis

The product of analysis from HOOD is an object model of a system.

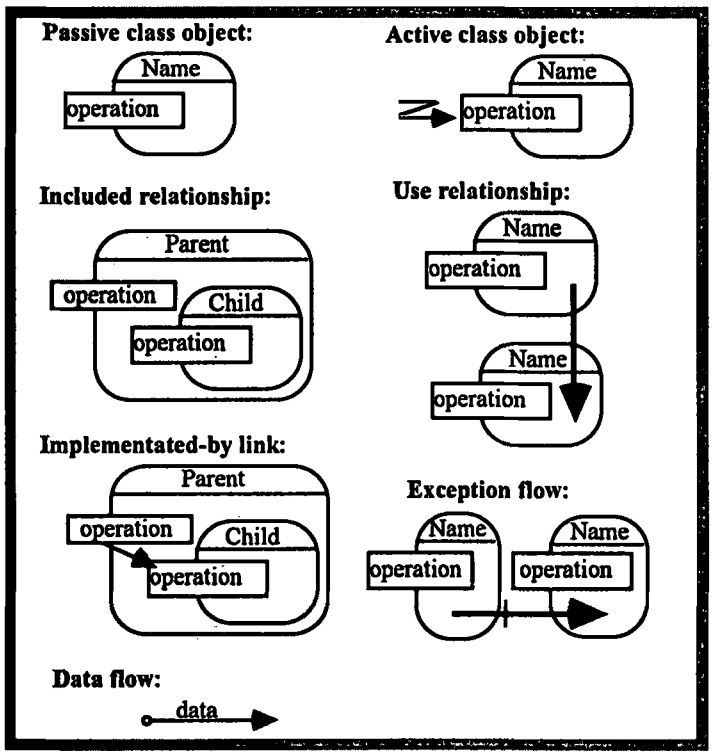

Figure 5.2 Symbols in HOOD Diagrams

References

[Abria80] Abria J.R., Schuman S. and Meyer B., A specification language, *On the Construction of Programs*, McNaughten R. and McKeag R. (eds.), Cambridge University Press.

[Ackroyd91] Ackroyd M. and Daum D., Graphical notation for object-oriented design and programming, *Journal Object-Oriented Programming*, January, 18-28.

[Alabiso88] Alabiso B., Transformation of data flow analysis models to object-oriented design, In *Proceedings of OOPSLA '88 Conference*, California, 335-353.

[Arnold91] Arnold P., Bodoff, S., Coleman, D., Gilchrist, H. and Hayes, F., Evaluation of five object-oriented development methods. *JOOP* (Focus on analysis and design), 101-121.

[Ashworth90] Ashworth C., *SSADM: A Practical Approach*, McGraw-Hill, London.

[Bailin89] Bailin S.C., An object-oriented requirements specification method. *Communication of the ACM*, 32(5), 608-623.

[Bear90] Bear S., Allen P., Coleman D. and Hayes F., Graphical specification of object oriented systems, In *Proceedings of ECOOP/OOPSLA 90 Conference*, 28-37.

[Booch86] Booch G., Object-oriented development, in *IEEE Transaction on software engineering*, 12(2), 211-221

[Booch91] Booch G., *Object-Oriented Design with Applications*, The Benjamin Cummings Publishing Company, CA.

[Byte81] Byte Special Issue: the Smalltalk-80 system, *Byte*, 6(8).

[Capretz93] Capretz, L.F. and Lee, P.A., Object-oriented design: guidelines and techniques, *Information and Software Technology*, 35(4), 195-206.

[Chen80] Chen P. (ed.), *Entity-Relationship Approach to Systems Analysis and Design*, North-Holland.

[Chen83] Chen P. (ed.), *Entity-Relationship Approach to Information Modelling and Analysis*, North-Holland.

[Coad91a] Coad P. and Yourdon E., *Object-Oriented Analysis*, Prentice Hall.

[Coad91b] Coad P. and Yourdon E., *Object-Oriented Design*, Prentice Hall.

[Coad91c] Coad P., OOA & OOD: a continuum of representation. *JOOP* , February, 55-56.

[Coleman91] Coleman D. and Hayes F. Lessons from Hewlett-Packard's experience of using object-oriented technology. *TOOLS 4*, Paris, 327-333.

[Coleman94] Coleman D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P., *Object-Oriented Development: The Fusion method*, Prentice Hall.

[Colter84] Colter M.A., A comparative examination of systems analysis techniques, *MIS Q*. March, 51-66.

[Constantine89] Constantine L., Object-oriented and structured methods: towards integration, *American Programmer*, 2(7), 34-40.

[Cook93] Cook S., Editorial—object-oriented systems, *The Computer Journal*, 32(4), 289.

[Cook94a] Cook S. and Daniels J., *Designing Object Systems: Object-Oriented Modelling with Syntropy*, Prentice Hall.

[Cook94b] Cook S. and Daniels J., Object-oriented methods and the great object myth. *Objects in Europe* , 1(4), Autumn, 13-18.

[Coutta87] Coutts, G., *SSADM—Structured Systems Analysis and Design Methodology*, Paradigm Publishing Ltd., London.

[Dahl78] Dahl O.J. and Nygaard K., Simular, an Algol-based simulation language, *Communication of ACM*, 9(9), 671-678.

[de Champeaux91] de Champeaux D., Object-oriented analysis and top-down software development. *LNCS 512 ECOOP 91*, 360-376.

[de Champeaux92] de Champeaux D. and Faure P., A comparative study of object-oriented analysis methods. *Journal Object-Orietned Programming*, 5(1), 21-33.

[DeMarco78] DeMarco T., *Structured Analysis and System Specification*, Yourdon Press, New York.

[Drake92] Drake J.M., Xie W and Tsai W.T., Document-driven analysis: description and formalization, *Journal of Object-Oriented Programming*, November/December, 33-50.

[Eckert94] Eckert G. and Golder P., Improving object-oriented analysis, *Information and Software Technology*, 36(2), 67-86.

[Embley92] Embley D.W., Kurtz, B.D. and Woodfield, S.N., *Object-Oriented Systems Analysis: a Model-Driven Approach*, Prentice Hall.

[Embley95] Embley D.W., Jackson R.B. and Woodfield, S.N., OO systems analysis: is it or isn't it?, *IEEE Software*, July, 19-33.

[Firesmith91] Firesmith, D., Structural analysis and object-oriented development are not compatible, *Ada Letters*, 11(6), 56-66.

[Fowler91] Fowler M., Choosing an object-oriented design method, *ECOOP 1991*, October, Seminar Notes.

[Fowler93] Fowler M., Object-oriented methods: a comparative overview, *SIGS Publication*,, 2-4.

[Freitas90] Freitas M.M., Moreira A. and Guerreiro P., Object-oriented requirements analysis in an Ada project, *Ada Letters*, Vol. 10, 97-109.

[Gane79] Gane C. and Sarson T., *Structured System Analysis: tools & techniques*, Prentice Hall.

[Gibson90] Gibson E., Objects—born and bred. *BYTE*, October, 245-254.

[Glykas93] Glykas, M., Whilhemij, P. and Holden, T., Object orientation in enterprise modelling and information system design. *Object-Oriented Development*, Colloquium organized by the British Computer Society Specialist Group on OOPS, January, 8/1-19.

[Graham91] Graham I., *Object-Oriented Methods*,, Addison-Wesley.

[Harel87] Harel D., Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8/231-274.

[Hayes87] Hayes I. (ed.), *Specification Case Studies*, Prentice Hall.

[Helm90] Helm, R., Holland, I.M. and Gangopadhyay, D., Contracts: specifying behavioral compositions in object-oriented systems, *ECOOP/OOPSLA '90 Proceedings*, 169-180.

[Henderson90] Henderson-Seller B. and Edwards J.M., The object-oriented systems life cycle, *Communication of ACM*, 33(5), 142-159.

[Henderson92] Henderson-Seller B., *A Book of Object-Oriented Knowledge—Object-Oriented Analysis, Design and Implementation: A New Approach to Software Engineering*, Prentice Hall.

[Honiden93] Honiden S, Kotaka N. and Kishimoto Y., Formalising specification modelling in OOA, *IEEE Software*, January, 54-66.

[IDE93] Interactive Development Environment Ltd. New IDE StP/MOT tools support entire software development processes, *Ref: IDE.025*, UK.

[Ince91] Ince D.C., *Object-Oriented Software Engineering with C++*, McGraw-Hill.

[ISO87] International Organisation for Standardisation, Concepts and terminology for the conceptual schema and the information base, *Ref. No. ISO/TR 9007: 1987 (E)*, Switzerland, July.

[Jacobson87]] Jacobson I., Object-oriented development in an industrial environment, *Proceedings of OOPSLA '87*, SIGPLAN Notices, 22(12), 183-191.

[Jacobson92] Jacobson B., *Object-Oriented Software Engineering—A Use Case Driven Approach*, Addison-Wesley.

[Jamsa84] Jamsa K.A., Object oriented design vs structured design—a student's perspective, *SIGSOFT Software Engineering Notes*, 9(1), 43-49.

[Kennedy88] Kennedy A.S. and Carter C.B., Structured analysis and structured design with Ada—a pragmatic object-oriented approach, *Ada User*, December.

[Kirk90] Kirk B.R., Designing Systems with Objects, Processes and Modules, *Proceedings of SE90* (Hall P. Ed.), Cambridge, 387-404.

[Lee91] Lee S. and Carver D.L., Object-oriented analysis and specification: A knowledge base approach, *Journal Object-Oriented Programming*, January, 35-43.

[Liang93] Liang, Y., Newton, M.A. and Robinson, H.M., Analysis of information systems using object-oriented methodologies, *Proceedings of 1st BCS & BSS Joint Conference on Theory, Use, and Integrative Aspects of IS Methodologies*, 57-70.

[Liang94] Liang, Y., Newton, M.A. and Robinson, H.M., The use of object models for information systems analysis, *ISD'94*, Bled, September.

[Lovegrove92] Lovegrove, G., Teaching an object-oriented design method, Strffordshire University.

[Martin92a] Martin J. and Odell J.J., *Object-Oriented Analysis and Design*, Prentice Hall.

[Martin92b] Martin J., *Principles of Object-Oriented Analysis and Design*, Published by Savant Institute, June.

[Meyer88] Meyer B., *Object-Oriented Software Construction*, Prentice Hall.

[Monarchi92] Monarchi, D.E. and Puhr, G.I., A research typology for object-oriented analysis and design. *Communication of ACM*, 35(9), 35-47.

[Nerson92] Nerson J., Applying object-oriented analysis and design, *Communication of ACM*, 35(9), 63-74.

[Olle82] Olle, T.W., Sol, H.G. and Verrijn-Stuart, A.A. (eds), *Information Systems Design Methodologies: A Comparative Review*, North-Holland.

[Olle83] Olle, T.W., Sol, H.G. and Tully, C.J. (eds), *Information Systems Design Methodologies: A Feature Analysis*, North-Holland.

[Olle88] Olle, T.W. Hagelstein, J., Macdonald, I.G., Rolland, C., Sol, H.H., Van Assche, F.J.M. and Verrijn-Stuart, A.A., *Information Systems Methodologies—A Framework for Understanding*, Addison-Wesley.

[Oxford88] Hornby A. S., *Oxford Advanced Learner's Dictionary of Current English* (fourth impression), Oxford University Press.

[Page89] Page-Jones, M. and Weiss, S., Synthesis: an object-oriented analysis and design method, *American Programmer*, 2(8), 64-67.

[Pressman87] Pressman R.S., *Software Engineering: A Practioner's Approach* (second ed.), McGraw-Hill.

[Robinson93] Robinson P.J., *Hierarchical Object-Oriented Design*, Prentice Hall.

[Rubin92] Rubin, K.S. and Goldberg, A., Object Behaviour Analysis, *Communication of the ACM*, 48-62.

[Rumbaugh91] Rumbaugh, J., Premerlani, J., Eddy, M. and Lorensen, W., *Object-Oriented Modeling and Design*, Prentice Hall.

[Rumbaugh94] Rumbaugh J., The life of an object model— how the object model changes during development, *Journal of Object-Oriented Programming*, 1.7(1), 24-33.

[Salmons93] Salmons J. and Babistky T., The computing world after objects, The International OOP Directory.

[Shelton93] Shelton R., Object-oriented analysis and design, *SIGS Publication*, 1.

[Shlaer88] Shlaer S. and Mellor S.J., *Object-Oriented System Analysis: Modelling the World in Data*, Prentice Hall.

[Shlaer92] Shlaer S. and Mellor S.J., *Object Lifecycles: Modelling the World in States*, Prentice Hall.

[Shumate91] Shumate, K., Structural analysis and object-oriented design are compatible, *Ada Letters*, 11(6), 78-90.

[Stroustrup86] Stroustrup B., *The C++ Programming Languages*, Addison-Wesley.

[Sully90] Sully P.D., and Ince D.C., The synthesis of object-oriented designs from the products of structured analysis, *SE90*.

[Sully93] Sully P., *Modelling the World in Objects*, Prentice Hall.

[Synder93] Synder A., The essence of objects: concepts and terms, *IEEE Software*, January, 31-42.

[Ten89] Ten Dyke R.P. and Kunz D.Q., Object-oriented programming, *IBM systems Journal*, 28(3).

[Verrijn82] Verrijn-Stuart A., CRIS: an introduction, *[also Olle82]*.

[Walker92] Walker. I.J., Requirements of an object-oriented design method, *Software Engineering Journal*, March, 102-113.

[Ward85] Ward P. and Mellor S., *Structured Development for Real-Time Systems*, Vol.1-3, Yourdon Press.

[Ward89] Ward P.T., How to integrate object orientation with structured analysis and design, *IEEE Software*, March, 74-82.

[Wasserman89] Wasserman A.I., Pircher P.A. and Muller R.J., The object-oriented structured design notation for software design representation, *IEEE Computer*, March, 50-62.

[Wiener88] Wiener R.S. and Pinson L.J., *An Introduction to Object-Oriented Programming and C++*, Addition-Wesley.

[Wiener91] Wiener R.S. and Friedman R.P., Foreward. *JOOP* (Focus on analysis and design), 4-5.

[Wirfs89] Wirfs-Brock, R. and Wilkerson, B., Object-oriented design: a responsibility-driven approach. *ECOOP/OOPSLA '89 Proceedings*, 71-75.

[Wirfs90] Wirfs-Brock, R.J., Wilkerson, B. and Wiener, L., *Designing Object-Oriented Software*, Prentice Hall.

[Yourdon89] Yourdon E., *Modern Structured Analysis*, Prentice Hall.